

المؤسسة العامة للدراسات والنشر والتوزيع



دومينيك لو فران

لغة آدا المساعد في البرمجة

ترجمة الدكتور عبد الحسن الحسيني



لغة آحا
المساعد في البرمجة

جميع الحقوق محفوظة
الطبعة الأولى
1408 هـ - 1988 م

 مؤسسة الجامعة للدراسات والنشر

بيروت - الحسنة - شارع النيل رقم ١٠ - بناية ١٠٠

هاتف : ٨٠٢٤٦٨ - ٨٠٢٤٦٧ - ٨٠٢٤٦٦

بيروت - الصفاة - بناية طاهر - هاتف : ٣١١٣١٠ - ٣٠١٠٣٠

ص. ب. ١٣٦١ / ١١٣ - تلکس : ٢٠٦٦٥٤٤ - ٢٠٦٦٥٤٠ - لبنان

سلسلة بإشراف
د. عبد الحسن الحسيني

دومينيك لو فران

لغة آدا المساعد في البرمجة

ترجمة الدكتور عبد الحسن الحسيني

المؤسسة الجامعية للدراسات والنشر والتوزيع

هذا الكتاب ترجمة :

**Le Langage
ADA
Manuel d'évaluation
Par
Dominique LE VERRAND**

© BORDAS, PARIS

الفصل الأول

مدخل

إن ظهور لغة جديدة تتمتع ببعض الأهمية للبرمجة لا يمكن أن يمر هكذا بدون أية ملاحظة في عالم المعلوماتية . لذا ، فجميع اللغات (ومن الصعب تعدادها) لم تعرف إنتشاراً كبيراً ، وفقط عدد صغير منها - إما لوسع إنتشارها ، وإما بسبب جذورها ، ومقدرتها - ترك أثراً في هذا العلم : FORTRAN ، Algol 60 ، Cobol ، APL ، Pascal ، Algol 68 ، Simula 67 ، PL/ 1 ، Basic ، Lisp .

لغة آدا (ADA) هي لغة جديدة للبرمجة ، ظهرت بدافع من وزارة الدفاع في الولايات المتحدة (DOD) . ومجالها ليس حصراً على التطبيقات والأعمال العسكرية ، ولكنها موجهة نحو كثير من التطبيقات المدنية : حساب علمي ، تطبيقات إدارية ، مناهج أساسية ، أنظمة في الوقت الفعلي ، مناهج واسعة الانتشار ، الخ . ظهور هذه اللغة وصناعتها . تطلب عملاً مشتركاً وشاقاً واحتاج لكثير من الاختصاصيين الدوليين من عالم البحث والصناعة [Rault 79, Wegner 80] .

بعض المراقبين يؤكدون دون لف أو دوران : إن لغة آدا ستكون لغة البرمجة للسنوات العشر 1990-1980 ، وهي ستزيح اللغات الأخرى . البعض الآخر ، يرى إن آدا هي لغة غير موفقة ، فهي من جهة الصناعة الحالية تفتقد إلى « الاسقاط العامودي Orthogonalité » ، وتعريفها غير الزامي بشكل خاص ، الخ . البعض الآخر ، يخشى من تعقيدها (يلزم أكثر من ثلاثة أشهر لمهندس ذي خبرة كي يفهمها) ، البعض الآخر يعتقد أنها غير كافية . في جميع الأحوال ، فأهمية الوسائط الموضوعة في العمل تؤدي إلى التكهّن بأن هذه اللغة ستكون من اللغات الكبرى المذكورة سابقاً ، وبشكل مستقل عن قيمتها الباطنية .

(1) انظر للمصفحات 1.4

(2) لغة كوبرول ظهرت أيضاً عن DoD ، وحتى الآن لا نزال ندرك أهمية هذه اللغة .

وللأسف ، فإن « ضجيجاً » وآراء خارجية ظهرت بسرعة ، وذلك بسبب فقدان المعلومات . وفي هذا المضمار وللمساعدة الاختصاصيين لبلورة أفكارهم قام فريق عمل بعرض وتطوير لغة Algol 68 . في نفس الإتجاه ننشر اليوم هذا العرض عن لغة ADA .

1.1 - غاية الكتاب

الأهداف :

بعد استيعاب الأهمية المحتملة للغة آدا خلال السنوات القادمة ، طلب مجموعة من الاختصاصيين في حقل المعلوماتية التعرف على هذه اللغة . ويدّوا بطرح السؤال التالي : « هل من الواجب اليوم ، أن نقوم بتوظيف المال في سبيل التعليم ، وغداً للحصول على المعرف ، لكي نستعمل معه ADA ؟ » هذا السؤال أخذ يُطرح بشكل حاد بالنسبة للمُصنّفين : « هل يجب توظيف الأموال في سبيل كتابة مصرّف ADA ؟ » . الجواب هو جزئياً « سياسي » ، ولكنه أيضاً تقني ، وهذا هو الرهان الذي يبدو أن البعض سيلعبه للتعرف عملياً على هذه اللغة .

ولتعلم لغة آدا ، يُهيّز الاختصاصيين اليوم بوثائق رسمية ، ودورات تدريبية منتشرة أكثر فأكثر ، وكتب تفصيلية بدأت بالظهور . وفي أغلب الأحيان لا تُؤلف هذه الوسائط سوى معلومات وصفية غير حرجة لامكانيات هذه اللغة .

وعلى العكس ، فالموضوع الأساسي لهذا الكتاب هو مساعدة الأشخاص المهتمين بلغة Ada ، لتكوين رأي يرتكز على معرفة تقنية وانتقادية لمختلف إتجاهاتها . كل مستعمل لأي لغة للبرمجة يُعتبر مستهلك ويعمل على هذا الأساس . هكذا ، فنحن « كمجموعة مُستهلكين » نعرض على أخواننا المعلوماتيين هذا الكتاب .

المستوى المفروض للقارئ

التقييم لا يعني أبداً إعطاء الرأي في هذا التصور أو ذاك . كي يبدو مفهوماً ، يجب أن يحتوي نص كهذا على قسم وصفي . وفي جميع الحالات ، هذا الكتاب ليس وثيقة تربوية : نفترض أن القارئ على معرفة مسبقة ببعض جوانب هذه اللغة وبشكل خاص بالمساعد المرجعي . أما القارئ المبتدئ فيمكن أن يقرأ بعض المداخل لهذه اللغة (مثلاً) ، وبعد ذلك يدرس المراجع بشكل متواز مع قراءة هذا العرض . إضافة لذلك ، فمعرفة جيدة بلغات البرمجة - على الأقل في مستوى لغة المعلوماتية الفرنسية - هي ضرورية لفهم النقدي أو التقني .

تصور نقدي

الاعلانات التي عرضناها لا تُؤلف « ما يجب أن يفكر بهذه اللغة » . ومن المعلوم أن الفائدة أو السيئة ليست مطلقة ، ويجب تلطيفها بعكسها وبالنظر إليها من خلال مفهوم

معيّن . إضافة لذلك ، وعلى عكس مجموعة العمل التي قامت بتقييم لغة Algol 68 ، لم نقم هنا بالبحث عن إجماع آراء مجموعتنا أو فريق العمل الخاص بنا حول هذه النقطة أو تلك . والنشأت التي جرى الإشارة إليها غايتها مساعدة القارئ على تكوين حكمه الشخصي عليها . حسب حقول إهتمامه . هكذا ، لم يبد لنا ضرورياً تبيان بعض النقاط الإيجابية والتي هي في هذه الأيام كلاسيكية . ولهذا السبب نرى إن بعض الفقرات تغطي إيجابياً على بعض المفاهيم السلبية .

من جهة أخرى ، لسنا مهتمين لتطابق اللغة مع دفتر شروطها [Steelman 77] . وأحكامنا تنطبق على اللغة نفسها ، كما فهمناها ، ودون أن نهتم بمعرفة الأسباب الإكسترا- معلوماتية التي جعلت المؤلفين (تلك الخاصة بدفتر الشروط ، وأعضاء فريق الإنشاء الذي يقوده Jean Ichbiah ، والواردة أسماؤهم في المساعد المرجعي) يعتمدون هذه الميزة أكثر من الأخرى . فمعرفتنا بهذه اللغة تركز بشكل أساسي على دراسة الوثائق الرسمية (أنظر 1.3) . هكذا ، فمن المحتمل أن نخطيء بالتفسير ، وذلك بسبب ضعفنا في بعض الأحيان ، ولكن أيضاً بسبب عدم الدقة ، أو بسبب الأخطاء الموجودة في المساعد المرجعي المستعمل [MR] . ولقد جرى توضيح أفكارنا بواسطة المناقشات والتناسب مع أعضاء فريق العمل التابع لجان إيشباه (J. Ichbiah) ، بشكل تكون فيه بعض أحكامنا خاصة بتنقيح الوثائق المستعملة .

2.2 - مجموعة وطريقة العمل

جرى تأسيس هذه المجموعة في الفصل الثاني من سنة 1980 في إطار أعمال فريق العمل GROPLAN (GROUpe Programmation et LANGages) للأقسام TTI و MA للمنظمة AFCET ، بواسطة جاك أندريه (Jacques André) (IRISA-Rennes) وروجيه روسو (Iman-NICE) . هذه المجموعة هي إعلامية وأعضاؤها ليسوا الممثلين المميزين لإداراتهم . ولم تحصل هذه المجموعة على أي مدة رسمية ، والفعل الحالي لم ينتج إلا من مبادرة خاصة . وتتألف من متقنين ، معاونين وقراء . اشترك الأوائل في تنقيح الكتاب الحالي . معهم ، قام الآخرون بالقراءة ونقد الصيغ المتتالية . والقسم الثالث منهم تابع عملنا من بعيد وأسماؤهم مذكورة لاحقاً في لائحة الشكر .

المتقنين والمعاونون إجتمعوا عدة مرات : أكتوبر 80 في نيس ، كانون الثاني 81 في أورو (Auron) ، آذار 81 في باريس ، حزيران 81 في Pouliguen ، تشرين 81 في Collioure . في هذه المناسبات ، وفي كل مرة خلال عدة أيام ، جرى مناقشة مختلف الفصول ، ونقدها وتصحيحها (بعد ذلك إعادة تصحيحها ونشرها في الاجتماع التالي) . فهذا العمل الذي نعرضه هو عمل جماعي ، حتى ولو كان كل فصل موقعاً ومنقحاً بفكر نقدي يختلف حسب المؤلفين .

1.3 - الوثائق المستعملة

تحتوي المراجع المذكورة في نهاية الكتاب على جميع المراجع الظاهرة في النص .
اللغات هي مذكورة بأسمائها والمنشورات بواسطة مفتاح بالشكل [nom (s) année] .
المفاتيح [MR] أو [MRA] ، [DF] ، [ME] و [GI] وتعني الوثائق « الرسمية »
للغة آدا ، لمعرفة :
- المساعد المرجعي [MR] و chapter reviews للنموذج المستقبلي [MRA] ANSI .
- مساعد الشروحات [ME] .
- إضافة إلى التعريف الشكلي [DF] والمساعد في العمل [GI] .

بالنسبة لـ [MR] ، فلقد اعتمدنا بشكل أساسي صيغة تموز 1980 لأنها الأخيرة التي
جرى نشرها في نفس الوقت الذي قدمنا فيه مخطوطتنا للتنقيح . هكذا أخذنا بالاعتبار
المراجعات التي جرت ابتداءً من تشرين أول سنة 1981 حتى نهاية نيسان 1982 ، من خلال
الملاحظات الداخلية (chapter reviews) التي أعلمنا بها Jean Ichbiah ، وبالتحديد
للفصول الأكثر تأثراً (7 و 13) . هكذا ، وفي العمق ، هذا التطوير يتطابق مع الصيغة
الجديدة الرسمية للغة آدا المذكورة «ANSI Standard Ada» [MRA] حيث المنشورات
معتمدة في نفس الوقت مع هذا الكتاب . هذا التوافق في الظهور يجعل هذا الكتاب غير
متطابق في جميع تفاصيله مع المعيار ANSI ، والوثائق التي نعمل عليها لها جانب مؤقت .
لن نقوم بالإشارة إلى الصيغة المسماة Green [79 Ichbiah] سوى بشكل استثنائي
وسنستعمل غالباً الصيغة [ME] ، التي تناسب GREEN ، لأنه لا يوجد صيغة إستيفاء
يومي من خلالها . أما بالنسبة لـ [DE] و [GI] فلن نقوم إلا بذكرها (انظر 14.1.6)
(7) . دون الدخول في تفاصيلها .

1.4 تحضير هذا العرض

تركيب هذا العرض

هذا العرض يركز على متابعة نفس البرنامج كالمساعد المرجعي ، لتسهيل العودة
إليه . وفي أغلب الأحيان ، سنلاحظ الفروقات التالية :
- الأنواع الرقمية تُعالج في فصل خاص ، لأنها لا تهم جميع المستخدمين .
- المناهج الثانوية والرمز جُمعت في نفس الفصل ، هذه الإنشاءات تتعلق بنفس المفهوم
التركيب .

(1) للعودة إلى الفصل C ، القسم S ، فقرة P من المساعد المرجعي سيتم بالشكل [MRC'S.P] [MRA]
ستتم الإشارة إليه بنفس الطريقة ، ولكنه لن يُذكر إلا حيث يختلف عن [MR] ، بينما العودة إلى الكتاب
الحالي سيتم بالشكل (انظر C.S.P) .

- إضافة لذلك ، فالفصل 12 يجمع جميع المفاهيم المناسبة للتكييف ، أي تلك الخاصة بالفصل 13 من [MR] ، والخواص والدرائج .

- الفصل 14 الذي يُعالج العناصر التشكيلية والنحوية والنصية ، موضوعي النهاية ، لأنه يقوم بمراجعة أمثلة متعددة أو إنشاءات موضوعية في الفصول السابقة .

تختلف فصول هذا العرض تتطرق إلى المفاهيم الأكثر عمومية . أما بالنسبة للمفاهيم الكلاسيكية فلم نعتبر مهماً التوسع طويلاً في عرض فائدتها والفصول المناسبة لها قصيرة . أما بالنسبة للمفاهيم الجديدة (كالعموميات ، الأعمال ، الشواذات ، الخ) ، فلندخل يُعبر عن فائدتها ويُراجع حالة العمل الحالية ، الفصول الخاصة بها هي غالباً طويلة .

الترجمة

لم يكن لدى مجموعتنا نزعة نحو المعايير . ولم ننشغل إذا بترجمة لغة Ada (لا كلماتها المحفوظة ، ولا تعريفها الإلزامي وغير الإلزامي) : ستكلم إذا عن الكلمة «USB» ، و ن التعليمية «IF» أو الخاصة DIGITS ، وسنحتفظ بالنحو في شكله الإنكليزي .

وعلى العكس ، فالفرنسية هي لغتنا الوطنية ، وسنجهد لاستعمالها بشكل صحيح (هنا سنجهد لاستعمال العربية بشكل صحيح لا يسيء إلى المعنى العام لتعليمات اللغة) . هكذا حاولنا ترجمة مفاهيم أدا بالمحافظة على الدلالة وعلى بعض التقريرات عندما نقدر على ذلك سنستعمل مثلاً بعض (pseudonomie) «أشباه الأسماء» لـ «aliasing» و «package» لـ package . وعلى العكس ترجمنا مثلاً embedded بواسطة «embôité» وليس بواسطة «imbriqué» وفي العربية سنستعمل الكلمة « متداخلة » للإشارة إلى نفس المفهوم .

سيجد القارئ في الملحق بعض المصطلحات الأساسية في هذا المساعد ، إضافة إلى ترجمة فرنسية للمصطلحات الإنكليزية المناسبة . وبالنسبة للنسخة العربية سنستعمل بعض المصطلحات العربية الشائعة .

المصطلحات

ستتبع مصطلحات [MR] ، بإمكان القارئ أن يستشير ملحق هذا الأخير . وفي بعض الأحيان ، تستأثر بعض المصطلحات الشائعة والمستعملة كثيراً في هذا الكتاب تعريفاً خاصاً .

صيغة الإسقاط العامودي (orthogonalité) جرى إدخالها بواسطة لغة ALGOL 68 . اللغة هي عامودية (orthogonal) إذا كانت أغلب إنشاءاتها- معبرة كمؤثرات - صالحة للجمع مع الإنشاءات الأخرى لهذه اللغة . تطبيق هذه الصيغة تُسهّل هذه اللغة وتُخفّف عدد المفاهيم الموضوعية في العمل ، والقواعد والإستثناءات ، ولكن هذا يؤدي عادة إلى تجميع للفوائد المُشكك بها ، وإلى صعوبات في الإنشاء أو إلى تعابير باطنية .

هكذا ، على سبيل المثال ، يجب على اللغة العامودية أن تسمح بإنشاء جداول دوال أو دوال تؤدي إلى نتائج من نوع جداول ، ولكن لغة كهذه ستسمح أيضاً بإنشاء سجلات عمليات أو مؤشرات .

المصطلح «illégal» أو «مغلوط erroné» له في هذا العرض معنى محدد كما في المساعد المرجعي [MR] . وقد يرتكب المبرمج أشكالاً مختلفة من الأخطاء عندما يكتب منهاجاً . ولغة آدا لا تتم إلا بثلاثة أنواع منها : البرنامج هو illégal إذا كان مرفوضاً ، في لحظة تصريفه ، بواسطة أي مصرف «عادي» للغة آدا (عادي يعني مطابق للنموذج أو للمعيار) .

بعض الأخطاء يجب أن تُكتشف في لحظة تنفيذ البرنامج ، بواسطة أي مُنفذ «طبيعي» . الأولوية المستعملة تدعى إستخراج «الاستثناء exception» ، وفي أغلب الأحيان تستعمل للسهر على الحوادث الشاذة المتوقعة بواسطة المبرمج .

يقال إن البرنامج هو مغلوط ، إذا خرج عن قواعد لغة آدا ، وإذا لم يكشف ذلك بواسطة المُصرِّفات والمنفذ . هكذا برنامج لا يحافظ على «فكرة اللغة» ويمكن أن يكون له أداء مختلف حسب برنامج التصريف والتنفيذ المستعملة .

طوبوغرافيا

من جهة الطوبوغرافيا ، نحن ملزمون باتباع الطوبوغرافية الموجودة في [MR] : تُكتب البرامج بلغة آدا باستعمال الأحرف السوداء لتمييز الكلمات - المفاتيح (مثلاً begin) ، والأحرف الكبيرة للمعرفات (INTEGER, BLANC) والأحرف الرومانية العادية للملاحظات (باستعمال مجموعة السمات الفرنسية ٤, ٥, ٦) . النحو هورومالي ، ولكن بالشكل الإنكليزي ، وفي النص ، الإستمعال المختلط للأحرف السوداء ، الكبيرة والإنكليزية يجب أن يؤدي إلى إلغاء أي إبهام (يجب أن لا نخلط الرقم مع الشكل النحوي «digit» أو مع الخاصية «DIGITS» أو الموصفة «digits») . فلنشر أيضاً إلى استعمال مجموعة سمات سوداء (تختلف عن تلك المستعملة للكلمات المفاتيح) للإشارة إلى بعض التعريفات أو إلى العناوين مما يسمح مثلاً بكتابة النوع type .

1.5 شكر

نتوجّه بشكرنا إلى :

- وكالة المعلوماتية (Agence de l'informatique) التي ساعدتنا مالياً لجمع وتصوير وطباعة هذا العرض .

- القسم TTI في AFCET ، والذي ساهم بدون مقابل بنشر الصيغة الأولى من هذا العرض .

- مختلف الادارات والمُنقّحين الذين ساهموا بمساعدة غير قليلة في السكرتاريا ، ويشكل خاص أولئك الذين نظموا العمل .

- « القراء » الذين ساهموا بعمل ملحوظ في نقد صيغ هذا العمل ، ويشكل خاص السادة « GRIZE ، Bourdel (CEA) ، Bondeli (CR2A) (جامعة Neufchatel) ، (GIXI) Guillon ، (EMD) Lemarche ، Schiper (جامعة لوزان) و(RNUR) Vojnor .

- جان إيشبياه Jean Ichibiah وفريق عمله ALSYS ، ويشكل خاص René Beretz ، الذين ساعدونا في هذا العمل بتقديمهم لنا الوثائق (Chapter reviews تحديداً) ، الشروحات والتشجيع .

- جاك أنثريه وزملائه في IRISA ، Jegou YVON وروبرت رانو ، الذين وبمساعدة مارتين جليفيو (Martin Glévéo) ، قاموا بتصوير وتشكيل وصنع ماكيت جميع صيغ هذا الكتاب .

الفصل الثاني

التصريحات والأنواع Declarations et types

رأبمه : Jean-Marie RIGAUD, université de toulouse

عملية التصريح تربط المرفء بوحدة ما في الفصل الأول من هذا الكتاب ، سنهتم بالتصريحات عن الأغراض ، الأعداد والأنواع . وفي القسم الثاني ، سندرس مختلف وسائط هذه اللغة التي تسمح للمستعمل بتعريف الأنواع . جميع هذه الصبغ جرى تطويرها في [MR3] ، [ME4] . وسندرس بشكل منفصل (الفصل 3) التصريحات من النوع رقمي [MR 3.5.6 + 3.5.9] .

2.1 - التصريحات

صناعة التصريح هي العملية التي يأخذ بواسطتها التصريح فعله . وبشكل عام ، هذه الصناعة تمر خلال دوران تنفيذ البرنامج . وتتألف من إنشاء الوحدات المصرح عنها إضافة إلى تفصيل شروط ، وإلزاميات هذا الإنشاء (مثلاً : تخصيص مكان من الذاكرة ، تطوير القيمة الأولية ، الخ) .

2.1.1 - التصريحات عن المواضيع (الأغراض)

2.1.1.1 العرض

التصريح عن الموضوع أو الغرض يؤدي إلى إدخال وحدة بسميزات نوع معين . يمكن أن تكون المواضيع عبارة عن متحولات أو ثوابت . وفي النهاية ، التصريح عن الغرض يمكن أن يكون متبوعاً بواسطة تعبير حيث التقييم يؤدي إلى إنتاج القيمة الأولية للغرض . مثلاً :

التصريح عن أغراض أو أعداد متحولة :

```
SOMME : INTEGER := 124 ;  
TEST1, TEST2 : BOOLEAN ;
```

تصريح عن أعداد (أغراض) ثابتة :

```
MAX : constant INTEGER := 100 ;  
LIMITE : constant INTEGER := SOMME*2/3 ;
```

2.1.1.2 - تقدير

إختبار النحو الخاص بالتصريح عن موضع أو عدد معين [MR 3.2] يؤدي إلى وجود نوع واحد ضمني مرادف (جدول) . لماذا هكذا إختيار ؟ . السبب سيؤدي إلى إلغاء الحاجة إلى تصريحات عن النوع في بعض البرامج . هذه البرامج يمكن ألا تستعمل سوى بعض الأعداد من نوع محدد سابقاً ، أو مواضيع جداول من نوع مرادف .
مهما يكن السبب ، فهذا يبقى صعباً ويذهب كاملاً إلى مواجهة الصفة العامودية للغة .

2.1.2 التصريحات عن الأعداد

2.1.2.1 التقدير

التصريح عن العدد يربط المعروف بقيمة رقمية ، هذه القيمة هي محدّدة بشكل تعبير حرفي حيث النوع مُستوحى من مركّبات التعبير .
مثلاً :

PI	: constant := 3.14 ;	- قيمة حقيقية
DIX	: constant := 10 ;	- قيمة صحيحة
DELTA	: constant := DIX/3 ;	- قيمة صحيحة 3

2.1.2.2 تطوير

المفاهيم المذكورة أعلاه (موضوع ثابت وعدد) تَبْلُو قِرية الواحدة من الأخرى .
وعملياً تختلف باستعمالها .

في التصريح عن الغرض أو الموضوع ، يجب أن يُحدّد النوع دائماً ومن الممكن التصريح عن المواضيع الثابتة من أي نوع . وعلى العكس ، فإن التصريح عن العدد لا يسمح إلا بالقيم الرقمية الصحيحة والحقيقية ، أي من النوع صحيح عام أو حقيقي عام .

من جهة أخرى ، فالتعابير المستعملة في التصريحات عن العدد هي تعابير حرفية ، أي ساكنة (أي قابلة للتعريف) . والأعداد المصرّح عنها يمكن أن تتداخل لاحقاً في التعريف عن أنواع أخرى حرفية . هذه الصفة لا تتحقق بواسطة الأغراض أو المواضيع الرقمية الثابتة .

● التصريح عن العدد يمكن أن يُفسّر كالتعريف عن تعبير جليد عن القيمة . في هذه الحالة ، الرمز « = » يبدو لنا وكأنه إختيار سييء ، ويُفضّل عنه الرمز « = » أو « := » .

2.1.3 - التصريحات عن الأنواع

2.1.3.1 - تقديم

يُميّز النوع ، مجموعة القيم ، التي يمكن أن تأخذها مجموعة الأغراض أو المواضيع أو الأشياء ، ومجموعة العمليات القابلة للتطبيق على هذه القيم . هذا المفهوم يسمح بتجميع خصائص المواضيع والأغراض مع التأكيد على أن هذه الخصائص لن تضيع في البرنامج الذي يُعالج هذه المواضيع . وهو يزيد من وضوح القراءة بتفصيل الانشاء التي يمكن أن تكون نموذجية أو معرفة بشكل منفصل .

بعض الأنواع هي معرفة أصلاً في هذه اللغة . التصريحات عن الأنواع تسمح للمستعمل بتعريف أنواع جديدة .

تعريف النوع - الثانوي من نوع معين يُضيق مجموعة القيم المقبولة دون تغيير في العمليات الممكنة .

يضاف عدة خاصيات إلى كل نوع وإلى كل نوع ثانوي ؛ ويقدم معلومات عن مميزات النوع أو الموضوع المحدد .

مثلاً :

- تصريحات عن الأنواع :

```
type INT is range 1..DIX ;  
type FRUIT is ( POMME, POIRE, PRUNE ) ;
```

- تصريح عن الأنواع الثانوية .

```
subtype SINT is INTEGER range 1..DIX ;
```

الخاصيات : لنفترض أن A هو موضوع أو غرض T هو نوع - ثانوي - قيمة بولية مُحدّد إذا كان الموضوع A'CONSTRAINED هو مُلزم أم لا .

T'FIRST للصفحة T'FIRST

من القيم المضافة إلى النوع - الثانوي T .

2.1.3.2 - تقدير

نجد في لغة آدا ، المفهوم العادي للنوع . وبالمقارنة مع لغة باسكال ، فإن لغة آدا تظهر بعض فجوات هذه الأخيرة (مثلاً : القيم الأولية ، الجداول الديناميكية) . ومن المؤسف أن بعض عمليات اختيار المفاهيم (أنواع مجهولة) تضرّ بصفة الإسقاط العامودي للغة .

2.1.4 - موقع التصريحات

يجب أن تجمّع التصريحات في الأقسام الوضعية [MR 3.9] . يوضع القسم

الوضعي في فقرة ، أو في برنامج ثانوي ، رزمة أو عمل . تُصنع التصريحات حسب الترتيب الواردة فيه في القسم الوصفي .

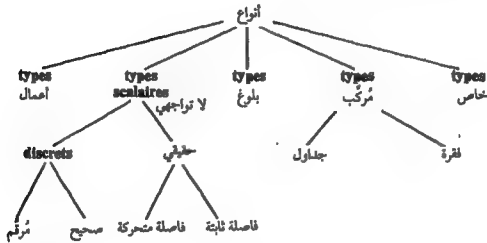
إنتباه :

يفرض النحو على التصريحات التي قد تُوصف « بالقصيرة » (موضوع ، نوع ، عدد ، إستثناء ، إعادة تسمية) أن تكون موضوعة قبل أية مواصفة تمثيل [MR 13.1] ، ولكن أيضاً ، قبل أي جسم للبرنامج الثانوي ، للرزمة أو للعمل .

السبب في إختيار كهذا بالنسبة للمؤلفين هو توضيح قراءة البرنامج .

2.2 الأنواع

2.2.1 تصنيف الأنواع حسب بنيتها .



لن نتكلم في هذا الفصل على الأنواع الخاصة ، فهي ستدرس في الفصل الذي سيعالج الأقسام (أنظر الفصل 6) . إضافة لذلك ، فالأنواع أعمال التي تُعرّف العمليات ستعالج في الفصل 8 ، والأنواع الحقيقية في الفصل 3 .

ملاحظة :

ما عدا النوع عمل ، فالأنواع المعرفة في هذه اللغة هي كلاسيكية . وبإمكاننا أن نتأسف على غياب النوع رزمة والنوع إجراء (procedure) بينما يدخل في اللغة النوع عمل . فلنلاحظ أيضاً غياب النوع «type» الذي لا يُشكّل إعاقة بالنسبة للمبرمج من جهة والذي يسمح بتوليد كود أكثر فعالية من جهة أخرى .

2.2.2 الأنواع المحلدة

هناك أنواع محلدة في كل إنشاء للغة :

INTEGER : صحيح

FLDAT : بفاصلة متحركة

DURATION : ثابت

CHARACTER, BOOLEAN : مُرَقَّم

STRING : جدول

NATURAL, PRIORITY : نانوي صحيح

هناك أنواع أخرى يمكن أن لا يتم تحديدها إلا في بعض الحالات :

LONG-INTEGER, SHORT-INTEGER : صحيح

LONG-FLOAT, SHORT-FLOAT : بفاصلة متحركة

2.2.3 طرق إنشاء الأنواع والأنواع - الثانوية

إضافة إلى الأنواع المحددة ، يمكن للمستعمل أن يُعرف أنواعاً جديدة . والوسائط

الموضوعة بتصريفهم هي :

● التقييم enumeration

● الاشتقاق مع أو بدون متطلبات .

● أدوات إنشاء الجداول ، الفقرات والبلوغ .

هذه الوسائط هي موضحة في الفقرة 2.3 .

ملاحظة

إن هذا الاختيار الذي برره مطوّلاً [ME4.2] يضمن وضوحاً أكبر في قراءة البرامج كما يضمن سهولة في التصريف .

2.2.5 التخصيص والمقارنة

يمكن إجراء عمليات التخصيص والمقارنة (التساوي والاختلاف) بين الأغراض لأي نوع من أنواع اللغة ADA (إلا في حال وجود مانع وضعه المبرمج بشكل ظاهر باستعماله أنواع خاصة محدودة) .

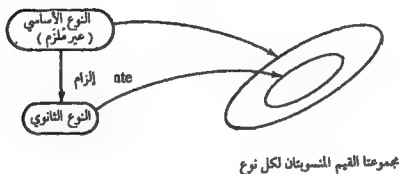
إن عملية التخصيص تستلزم أن يكون الغرضان (المرسل والمستقبل) من نفس النوع . إذ لا يوجد تغيير أو تحصيل ضمني للأنواع . وسندرس بالتفصيل في الفصل الرابع التغييرات الظاهرية .

2.2.6 الأنواع الثانوية

بالإمكان تحديد نوع ثانوي بواسطة إلزام معيّن لنوع نسمّيه النوع الأمامي للنوع الثانوي .

ويسمح هذا الإلزام بتقليص مجموعة القيم المنسوبة إلى النوع الأساسي ، وهو بإمكانه أن يكون :

- إلزاماً من ناحية المجال (أنواع لا اتجاهية Scalar)
- إلزاماً من ناحية الدلالة (الأنواع جداول)
- إلزاماً من ناحية المميز (discriminant) (الأنواع فقرات)
- إلزاماً من ناحية الدقة (الأنواع الحقيقية) .



مثلاً :

```
subtype COMPTEUR is INTEGER range 1..100 ;
subtype WEEKEND is JOUR range SAM..DIM ;
subtype INTEGER2 is INTEGER ; -- INTEGER
```

أنظر الفقرة 2.3.1 - هو مرادف لـ

إن التصريح عن نوع ثانوي لا يقوم بتحديد نوع جديد ، ولكن يقلص مجموعة القيم المقبولة من هذا النوع . كل غرض مصرح عنه تحت اسم هذا النوع الثانوي ينتمي إلى النوع الأساسي لهذا النوع الثانوي ولكنه يأخذ قيمة فقط في المجال الذي حدده الإلزام المنسوب إلى النوع الثانوي .

ملاحظة

يعتمد [MR] على توالي العبارتين « نوع » و « نوع ثانوي » . بشكل عام ، تستعمل العبارة « نوع ثانوي » عندما يريد المؤلف الإشارة إلى وجود (محتمل) للإلزامات . والعبارة « نوع » تدل عامة على الأنواع والأنواع الثانوية على السواء (مجموعة القيم + مجموعة العمليات) .

2.3 - إنشاء الأنواع

2.3.1 التعداد

يسمح التعداد بتحديد نوع لا اتجاهي (Scalar) بواسطة لائحة من القيم المؤلفة .

ويشار إلى هذه القيم بواسطة معرفات أو سمات حرفية .

مثلاً

```
type COULEUR is ( ROUGE, JAUNE, GRIS, VERT, BLANC );  
type JOUR is ( LUN, MAR, MER, JEU, VEN, SAM, DIM );  
type LUMIERE is ( JAUNE, ORANGE, BLANC );  
type CH_HEXA is ('0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F');  
type CH_ROMAIN is ('T','V','X','D','C','L','M');
```

إنّ التعداد يأخذ مجدداً خصائص الأنواع المملوءة في لغة PASCAL (الترتيب ، الموقع) . إضافة إلى هذا ، بإمكان قيمة معينة أن تظهر في أكثر من نوع بالتعداد ، وتدعى عندها قيمة يحمل زائد (Overloaded) . عند استعمال قيم كهذه ، قد يتوجب على المبرمج أن يحدد النوع الذي تنتمي إليه هذه القيم لا سيما عندما لا يكون بالإمكان استنتاج هذا النوع من الإطار العام ، ولهذا فإنه سيستعمل عبارة مميزة [MR4.7] .

مثلاً

```
for I in JAUNE..BLANC -- غير واضحة  
for I in COULEUR('JAUNE')..COULEUR('BLANC') -- واضحة
```

وذلك لأن JAUNE و BLANC ظهرا تباعاً في النوع COULEUR وفي النوع LUMIERE .

ملاحظة

إنّ إدخال قيم جديدة من النوع تعداد في برنامج موجود أصلاً بإمكانه أن يؤدي إلى زيادة تحميل البعض منها . من هنا ضرورة وضع عبارات مميزة لرفع بعض سوء الفهم وعدم الوضوح . وقد لا تكون التغيرات الناتجة وفقاً على الأجزاء الجديدة من البرنامج بل تتمدها إلى الأقسام الموجودة أصلاً . لكن لا ننسى أن أحد أهداف اللغة ADA هو تخفيض كلفة ضبط البرامج والمحافظة عليها .

2.3.2 الاشتقاق

بإمكاننا الاشتقاق انطلاقاً من أي نوع نسميه النوع القريب ، ويسمح الاشتقاق بتحديد نوع جديد هو النوع المشتق . ويتمتع هذا النوع بالخصائص الآتية :

- مجموعة قيم النوع المشتق هي نسخة عن مجموعة قيم النوع القريب .

2.2.4 - تعادل الأنواع

تعادل الأنواع يتم بالإسم .

مثلاً :

```

type T1 is
record
  C1 : INTEGER ;
  C2 : REAL ;
end record ;

```

هذان التصريحان يُدخلان أنواع مُحددة ومختلفة مع أنَّ مجموعاتهم قيمها وتركيباتها متشابهة .

```

type T2 is
record
  C1 : INTEGER ;
  C2 : REAL ;
end record ;

```

في لغة آدا ، نعتبر إن موضوعين هما من نفس النوعية :

- إذا كانا مرتبطين بنفس إسم النوع أو
- إذا ظهرا في نفس التصريح عن الجدول الملزم .
- العمليات الممكنة على الأنواع المشتقة هي :
- العمليات المحددة للنوع القريب إذا كان هذا الأخير من نوع محدد سابقاً .
- العمليات الممكنة على النوع القريب إذا كان هذا الأخير هو من نوع مشتق (أي إذا كانت العمليات بعد الاشتقاق) .
- العمليات المحددة في نفس مواصفة الرزمة كالنوع القريب إذا كان هذا الأخير مصرحاً عنه في مواصفة رزمة ويشترط أن يكون تعريف النوع المشتق يدخل بعد نهاية مواصفة الرزمة .

مخطط تعريف مواصفة النوع المشتق هو التالي :

type T2 is new T1 ;



مثلاً :

```

type LUMIERE_BIS is new COULEUR ;
type DISTANCE is new INTEGER ;

```

خواص النوع قريب يجري المحافظة عليها بواسطة النوع مشتق .
التحويل الخارجي هو ممكن بين قيم النوعين المشتقين (فلنشر إلى أن أي تحويل
ضعفي لا يوجد في لغة إذا بين قيم نوعين محددين .

ملاحظة :

الفقرة [MR 3.4] تعالج العمليات المشتقة عندما يكون اشتقاق النوع غير
واضح . التفسير المعطى هنا هو ذلك الخاص بـ [G1 3.4] .

2.3.2 - الاشتقاق مع الإلزام
من الممكن توحيد الإلزام والاشتقاق على الشكل التالي :

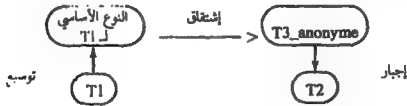
`type T2 is new T1 range INF..SUP ;`

هذه الصورة تعادل :

`type T3_anonyme is new type_de_base_de_T1 ;
subtype T2 is T3_anonyme range INF..SUP ;`

من المحتمل أن تكون T1 إسماً للنوع - الثانوي . والنوع الأساسي لـ T1 ، غير
الإجباري إذا ، هو قبل أي شيء مشتق من نوع مجهول (anonyme) ، هذا الأخير ، مع
الإلزام يعرف النوع - الثانوي .

إضافة لذلك ، فإذا كان T1 هو من نوع مُلزم ، فالإلزام المفروض على T2 يجب أن
يكون مُتَكيِّفاً مع النوع الخاص بـ T1 .



كتابة مبسطة في حالة العدد الصحيح تسمح بكتابة :

`type T1 is range INF..SUP ;`

في هذه الحالة ، يُختار النوع « قريب » بواسطة المصروف ، ومن الأنواع الصحيحة
المحددة ، كي يتم تطبيق الإلزام المحدد .

مثلاً :

```
type LUMIERE_TER is new COULEUR range GRIS..BLANC ;
type INDEX is range 1..1000 ;
```

2.3.4 تقدير

أ - الأنواع المشتقة والأنواع الثانوية هي وسائط مفتاح في لغة آدا . بعد تعريفها من خلال صيغة بسيطة (نسخة جديدة عن مجموعة خصائص ، تفصيل في حدود فسحة من القلم) ، هذه الوسائط تصبح أكثر فأكثر فعالية .

مثال 1

```
subtype LONGUEUR is INTEGER range 0..100 ;
subtype POIDS is INTEGER range 0..200 ;
... -- sous-types d'entiers
```

```
L : LONGUEUR ;
P : POIDS ;
```

```
begin
```

```
  L + P =
```

تعبير مسموح به ولكن بدون أي معنى فيزيائي
(التصريح عن النوع الثانوي لا يعرف النوع الجديد)

مثال رقم 2

```
type LONGUEUR is new INTEGER range 0..100 ;
type POIDS is new INTEGER range 0..200 ;
... -- LONGUEUR et POIDS sont deux types dérivés de INTEGER
L : LONGUEUR ;
P : POIDS ;
```

```
begin
```

```
  L + P =
```

هذا غير مسموح ويكتشف خلال التصريف
(التصريح عن النوع المشتق يؤدي إلى إدخال نوع جديد)

b - تُستعمل مُلزمات الحقل في التعريف عن النوع - الثانوي . هكذا إلزام يُجَلَد بواسطة تعبيرين بسيطين [MR 3.5] ويُشكّلان حدود الحقل .

بنينا في التصريح عن النوع الصحيح [MR 3.5.4] يجب على كل حد أن يكون عبارة عن تعبير ساكن ، لا يعطي أية دقة - بالنسبة للمحدود الأخرى ، ومن الممكن أن نستخلص إن هذه التعبيرات هي قابلة للتقدير عند التنفيذ .

c - تضاف الخواص إلى الأنوع المُجزّاة (discut) [MR 3.5 و MR 3.5.5] . من هذه الخواص ، بعضها (FIRST, LAST) يُربط بالنوع ، وإحتمالاً بالنوع الثانوي ، والبعض الآخر (POS, SUCC, PRED, VAL) يُربط بالنوع الأساسي .
مثلاً :

```
type T1 is (V0, V1, V2, V3, V4, V5, V6);
subtype ST1 is T1 range V2..V4;
```

```
- 1ère catégorie
  T1'FIRST    → V0
  T1'LAST     → V6
  ST1'FIRST   → V2
  ST1'LAST    → V4

- 2ème catégorie
  T1'SUCC (V4) → V5
  ST1'SUCC (V4) → V5
  T1'POS (V2)  → 2
  ST1'POS (V2) → 2
  ST1'POS (V5) → 5
```

بينما V5 لا تنتمي -
في حقل قيمة ST1 -
نفس الملاحظة -

2.3.5 - الجداول

في لغة أدا يمكن أن تُعرّف الجداول حسب طريقتين :

- جدول مع إلزام : جميع مواضيع (أعداد) النوع هي بنفس الحدود .
- جدول بدون إلزام . مختلف مواضيع النوع يمكن أن تكون بحدود مختلفة .

الجدول يمكن أن تكون بعدة أبعاد ويكون حدود مفروضة من اللغة .

2.3.5.1 - الجداول مع الإلزام (الجداول المحددة)

هذه الجداول هي كلاسيكية ، وهي نفسها التي نعرفها في أغلب اللغات (باسكال ، PL/1 ، Algol) مع نحو قد يختلف في بعض الأحيان .
مثلاً :

```
type TABLE is array (1..10) of COULEUR ;
```

```
- نوع جدول مع إلزام -
- مواضيع مصرّح عنها في النوع TABLE -
T1, T2 : TABLE ;
T3 : array (1..50) of TABLE ;
```

2.3.5.2 - الجداول بدون إلزام (الجداول غير المحددة)

هذه الجداول هي عبارة عن نماذج لا يتم فيها تحديد قيم حقول الدلائل بالكامل ،

هكذا مواصفات للجداول لا يمكن أن تستعمل إلا في التصريحات عن الأنواع .
مثلاً :

-- Rappel : type INDEX is range 1..1000 ;
type VECTEUR is array (INDEX range <>) of REAL ;
V1 : VECTEUR (1..10) ; تعريف عن الموضوع
subtype VECTEUR_50 is VECTEUR (1..50) ; تعريف عن النوع الثانوي
V50 : VECTEUR_50 ;

نُحلّد قيم الدليل في فسخة تدعى إلزام الدليل .
التصريح عن موضوع في نوع جدول بدون إلزام يجب أن يحتوي على إلزام
للدلائل . بنفس الطريقة ، فمن الممكن تعريف أنواع ثانوية لنوع جدول بدون إلزام .

2.3.5.3 - حدود إلزام الدليل
تُستعمل حدود إلزام الدليل في التصريحات عن الأنواع الثانوية أو المواضيع . النوع
الأساسي في التعريف عن نوع ثانوي والذي يستعمل حدود دليل يجب أن يكون من نوع
جدول بدون إلزام .

التعابير المستعملة في إلزام الدليل يمكن أن تكون ديناميكية ، أي قابلة للحساب عند
التنفيذ . هكذا جداول تدعى جداول ديناميكية .

عندما يُصرّح عن الموضوع بالخاصية « ثانية constant » ، ليس من الضروري
تحديد حدود إلزام الدليل ، هذا الإلزام يمكن أن نحصل عليه من القيمة الأولية
للموضوع . وفي الحالة التي يكون فيها الحد الأدنى أقل من إلزام الدليل وغير محدد ، يؤخذ
مساوي لـ S'FIRST إذا كانت S هي إسم النوع - الثانوي المستعمل في التعريف الجزئي
للإلزام .
مثلاً :

V_DYN : VECTEUR (N..N+30) ;
N - هو موضوع
V_CONS : constant VECTEUR := (4.0, 3.5, 2.1) ;
- إلزام الدليل لـ V-Const هو 1..3
لأن INDEX-FIRST = 1

- انتباه
type VECTEUR_INT is array (INTEGER range <>) of REAL ;
V_INT : constant VECTEUR_INT := (1.0, 2.0, 3.0) ;
- إلزام حدود الدليل V-INT ليس أكثر من 1..3
- الحد الأدنى يعادل INTEGER'FIRST
- (31 * 2 * مثلا) .

من الممكن تعريف جدول لا يحتوي على أي عنصر . يكفي لذلك أن يكون الحدُّ الأعلى هو السابق للحد الأدنى . هذه الأخيرة يجب أن تنتمي إلى إلزام حدود الدليل .
مثلاً :

V_NULL10 : VECTEUR (10..9); -- OK
V_NULL1 : VECTEUR (1 ..0); -- INDEX لا ينتمي أبداً إلى النوع
-- ولكن ، التدقيق يتم على 1 .
-- هذا التصريح هو صحيح .

أمثلة غير صحيحة .

V_NULL0 : VECTEUR (1001..1000);
-- 1001

V_NULL2 : VECTEUR (2..0); --
-- 0 لا ينتمي أبداً إلى النوع INDEX وليس سابقاً لـ 2 .
-- تغيير إبتداءً من [MRA] .

2.3.5.4 - ملامح السمات

النوع «STRING» الرمزي يُحدَّد بشكلٍ معادل للتصريح التالي :

type STRING is array (<NATURAL range>) of CHARACTER ;

العمليات المحددة على الجداول ذات البعد الواحد هي أيضاً صالحة للتطبيق على المواضيع من نوع STRING .
مثلاً :

DATE LIM : STRING (1..15) := "15SEPTEMBRE1981";
NOI1 : constant STRING := "25DEC80";
subtype TEXTE16 is STRING (1..16);

2.3.5.5 التقدير

● الجدول بدون إلزام يجب أن يُعتبر كنموذج للجدول الذي يبقى فيه متغير واحد للتعريف : حقل قيم الدلائل . هكذا مواصفة للجدول هي محدّدة بالتصريح عن النوع . كل تصريح عن الموضوع من خلال هذا النموذج يجب أن يحتوي على جميع حدود الإلزام عن الدليل غير المحدد .

الجدول الديناميكي هو موضوع من نوع جدول حيث على الأقل أحد حدوده غير قابل للتقدير عند إنشاء التصريح . مفهوم الجدول الديناميكي هو مختلف عن مفهوم الجدول

المرن في لغة Algol 68 ، في هذا المعنى كما في آدا لا يمكن تعديل الحدود بعد التصريح ، كما في Algol 60 .

● من النحر [MR 3.6] ، إذا كان الجدول بدون إلزام يحتوي على عدة أبعاد ، فجميع الأبعاد يجب أن تكون بدون إلزام .

● استعمال الجداول بدون إلزام كمتغير وسيطي شكلي في برنامج - ثانوي ، يسمح بإرسال متغيرات وسيطية من نفس نوع - جدول بحقول دلائل مختلفة . يجب لهذا أن تكون جميع المتغيرات الوسيطة الفعالة مصرحاً عنها من خلال نوع بدون إلزام كمتغير وسيطي شكلي .

مثلاً :

```
type TABLE is array ( INTEGER range<> ) of INTEGER ;
subtype TABLE10 is TABLE ( 1..10 ) ;
```

```
...
T1, T2 : TABLE10 ;           - تصريح عن المواضع
T3 : TABLE ( 50..100 ) ;
```

```
...
procedure TRI ( T : in out TABLE ) ; - تصريح عن برنامج ثانوي
```

```
...
procedure TRI ( T : in out TABLE ) is
```

```
...
begin
    ...
    for I in T'RANGE loop
```

```
...
end TRI ;
```

```
...
TRI ( T1 ) ;                   - نداء إلى البرامج الثانوية TRI
TRI ( T2 ) ;
TRI ( T3 ) ;
```

● النوع المحدد STRING يُمثل سلاسل السمات بطول ثابت . من هنا ، وعند تخصيص أداة من نوع STRING ، يجب أن يكون عدد السمات في هذه الأخيرة معادلاً لعدد العناصر في الموضوع المستقبل [MR 5.2.1] .

مثلاً :

```
DATE_LIM := "10_OCTOBRE_1981" ; -- OK
DATE_LIM := "10/10/81" ; -- OK
```

```
DATE_LIM := "10/10/81" ;      - غير مسموح
```

ولكن :

تأسفن على إمكانيات معالجة النص في اللغات COBOL

هو محمّد فلا مانع أبداً من التصريح التالي :

```
subtype TEXTE50 is STRING (50..100);
```

وع article :

(dis

نوع فقرة على قيم أولية بالغلط . هذه القيم هي مستعملة
صغير وإعداد ضمنية محدّدة في التصريح عن الموضوع .

س التركيب .

```
type DATE is
  record
    JOUR : INTEGER range 1..31 ;
    MOIS : STRING (1..10);
    AN   : INTEGER range 0..3000 :=
  end record ;
```

يُستعمل لـ :

مركّب (من نوع جدول) الفقرة .

من الفقرة .

```
type MAT_CARREE (DIM : INTEGER range
  record
    MAT : array (1..DIM, 1..DIM) of REA
  end record ;
```

- فقرة مع إلزام .

```

type T (DISCR : COULEUR) is
record
  T : INTEGER ;
  case DISCR is
    when ROUGE => K : CHARACTER ;
    when VERT  => L : INTEGER range 0..1000 ;
    when others => null ;
  end case ;
end record ;

```

2.3.6.3 المميزات

المُمَيِّز يجب أن يكون من نوع مجزأ *discret* . ويمكن أن يحتوي على قيمة أولية بالغلط .

القيمة المُمَيِّزة للموضوع من نوع فقرة لا يمكن أن تُعدّل إلا بتخصيص كامل للموضوع . إضافة لذلك ، فهذا التعديل هو غير ممكن إلا إذا كان الموضوع غير إلزامي .

الموضوع - فقرة هو إلزامي إذا جرى استعمال إلزام في المُمَيِّز :

- في التصريح عن الموضوع .

- في التصريح عن النوع - الثانوي من خلال تعريف الموضوع .

الموضوع هو بدون إلزام عندما تكون القيم بالغلط للإلزام في المُمَيِّز مُستعملة .

مثلاً :

```

S1 : MAT_CARREE ;          DIM = 16 فقرة غير ملزمة
S2 : MAT_CARREE (8) ;      - فقرة ملزمة
subtype ST is MAT_CARREE (8) ;
S3 : ST ;                  فقرة إلزام

```

القيمة الأولى هي قيمة المُمَيِّز غير صحيح ، S2 هو إلزام

```

S1 := (4, (1..4) => (1..4 => 0.0)) ;
S2 := S1 ;
S3 := (8, (1..8) => (1..8 => 1.0)) ; -- OK
S1 := S3 ;                          -- OK

```

2.3.6.4 البدائل

ال فقرات مع إلزام هي شبيهة لتلك الموجودة في لغة باسكال . منقاة البديل يجب أن تظهر كمُمَيِّز للفقرة .

منقاة الاختيار يجب أن تكون :

- تعبيراً بسيطاً وساكناً .

- فسحة من القيم .
- أي شيء آخر (others) ، في آخر مكان ، يعني جميع القيم الغير معدة .

2.3.6.5 تقدير

مُركب الفقرة يمكن أن يُعدّ بالغلط ، ومن الممكن أن نأسف إلى إن هذه الخصوصية هي غير مُعمّمة على مجموع الأنواع . ومن المحتمل أن نصل إلى نفس النتيجة بوضع النوع المعتمد في تسجيلة (record) ، وهذا فعلاً حلّ حقيقي .
يمكن أن نصرّح عن الفقرة بدون مركب (اللاتحة الفارغة تُحلّد بواسطة null) .
مثلاً :

```
type RIEN is
  record
    null ;
  end record ;
```

هذه الصفة هي عنصر مُساعد لاستعمال اللغة في توليد البرامج .
● مع إن هذا لا يظهر إلا في النحو ، فإن مركبات الجداول للفقرة يجب أن تكون مع إلزام .

● من غير الممكن إستعمال نفس أسماء المركبات في إختيارين مختلفين لقسم من البديل . وهذا قد يبدو مثيراً عند كتابة بعض الخوارزميات .

2.3.7 عمليات البلوغ

أنواع البلوغ في لغة آدا ، هي عبارة عن وسائط للبلوغ إلى أعداد أو مواضيع منشأة ديناميكياً وبمجهولة . قيم نوع البلوغ تعني المواضيع المجهولة .

يضاف نوع البلوغ عند التصريح عنه إلى -نوع معين . لا يمكن أن تعني قيم البلوغ المناسبة إلا مواضيع من هذا النوع . هكذا تقييد يسمح بتحديد كامل وساكن للأنواع .
مثلاً :

```
type AP is access PERSONNE ;
X, Y : AP ;
```

القيم Null لا تعني أي موضوع ، وتنتمي إلى جميع أنواع البلوغ . تُصفر وتُعد جميع مواضيع البلوغ مع هذه القيمة عند الإنشاء . القيم الأخرى نحصل عليها باستعمال (allocateur) .

النوع Type المضاف إلى النوع بلوغ Type access يمكن أن يكون نوع - جدول بدون إلزام أو نوع - فقرة مع تمييز . وفي كل حالة ، فإن الإلزام يجب أن يحدد في لحظة الانشاء ، والموضوع المخصص هو مع الإلزام .

النوع - بلوغ يسمح بتصريح تكراري للأنواع . لهذا ، فالتصريحات غير الفاصلة للنوع هي ممكنة .

مثلاً :

```
type ELEM ;                - تصريح غير كامل
type PTR is access ELEM ;  - متمم التصريح
type ELEM is
  record
    INFO : INTEGER ;
    SUCC, PRED : PTR ;
  end record ;
```

ملاحظات :

● في لغة آدا ، يُستعمل نوع - البلوغ لتحديد المواضيع الديناميكية . هكذا ، فقيمة البلوغ لا يمكن أبداً أن تعني موضوعاً ساكناً .

● يضاف المخصص إلى نوع الموضوع المؤشر وليس إلى موضوع - البلوغ كما في لغة باسكال .

2.4 خاتمة

مع مفهوم كلاسيكي للأنواع ، يبدو أن مصممي اللغة قد لحظوا بشكل أولي إمكانية العمل للبرمجة ، التقييد في تعديل تمييز الفقرات (ما عدا في الحالات الخاصة) ، ووجود الإلزام في الحقل ومفهوم النوع المشتق هي خيارات في هذا الاتجاه . وفي إطار هذه اللغات الموجهة إلى التطبيقات في « الوقت الحقيقي » ، يبدو أن آدا قد وجدت شيئاً مشتركاً بين التدقيق الساكن والديناميكي . عمليات التدقيق المضافة إلى المفاهيم نوع ونوع ثانوي يمكن أن تتم ، لأغلب اللغات ، عند التصريف .

هناك ظلّ حول الجدول : النحو الذي لا يبدو وكأنه جيد ووجود النوع المجهول للجدول ؛ الملاحظات الأخرى تبقى صغيرة .

الفصل الثالث

الأنواع الرقمية

3.1 - مدخل

مسألة الأنواع الرقمية في لغة للبرمجة هي مسألة التمثيل المحدد والمتقطع للمواضيع والأغراض التي تستطيع أن تأخذ مضموناً من القيم . لنرى أولاً لماذا يمثل هذا الأمر مشكلة . بعد ذلك سنعرض الأسس النظرية للحل المختار بواسطة مؤلفين آدا . سنشير في النهاية إلى بعض المواضيع المفتوحة .

النوع T هو المعطى من مجموعة T والعمليات (الموحدة ، الثنائية ، النخ) ، على هذه المجموعة . وبشكل عام ، من الممكن أن نعطي طبقة من الأنواع T1 والدوال بين عمليات الضرب الهرتزي لـ T1 مثلاً :

الأسية : $REEL \times ENTIER \rightarrow REEL$

صحيح \times حقيقي \leftarrow حقيقي

الجمع : $REEL \leftarrow REEL \times REEL$

حقيقي \times حقيقي \leftarrow حقيقي

يُعرف النوع T بواسطة معرف ، وإسمه ومن الممكن أن نضيف إليه عنصراً من T ، وقيمه . ولا تتمتع الكائنات إلا بعدد محدد ، كبير ، من الحالات الممكنة ، ولغة البرمجة لا تسمح بتعريف سوى عدد محدد من المواضيع وبالأخص نُحدد قيمها الممكنة بمجموعة منتهية محدودة . المشكلة نفرض إذا إضافة أنواع « غير محدودة » (كالأعداد الحقيقية ، والدوال ، النخ) ، إلى أنواع محدودة تمثلها بشكل محدد .

مثلاً : نضيف إلى الأعداد الحقيقية مجموعة الأعداد بفاصلة « متحركة » ، أي الأعداد القابلة للتمثيل بالتحديد على مكنة معينة ولكن ولتعريف النوع بفاصلة متحركة FLOTTANT ، يجب أن نُحدد له فعل العمليات + ، + ، النخ . أو إذا كانت $y \neq 0$ ، فهذا العدد يتعلّق (X) تعني العدد بفاصلة متحركة المضاف إلى نتيجة العملية $y \times x$ ،

بالطريقة التي تُمثّل بها x و y ، وبالطريقة التي يتم بها التلوين ، الخ . وبكلمة أخرى ، هناك النوع FLOTTANT لكل مكنة . وعدة أنواع ، إذا أخذنا بالحسبان إمكانيات التمثيل بالدقة المضاعفة . ولا يبدو أن محاولات المعالجة في تمثيل الأعداد سنصل إليها في وقت قصير ، كون الاتفاق يفترض الارادة الحسنة من قبل المصممين ، والمسألة ستؤدي إلى بعض المشاكل .

ليس هناك ما يبدو أنه من الممكن تعريف النوع FLOTTANT النموذجي العام .
(أنظر في هذا الموضوع [Stevenson 81] و [Cody 81]) .

حتى ولو استطعنا الوصول لذلك ، فلن يكون ممكناً الحصول على دلالة بسيطة . مثلاً ، من الممكن أن نُفضّل أن يكون العدد $f1(x * y)$ هو دائماً العدد بفاصلة متحركة الأقرب من العدد الحقيقي $x * y$. فإذا كانت الحالة كذلك ، فسيكون بإمكاننا أن نؤمن إن :

$$(3.1) \quad \Pi(x \otimes y) = (x \otimes y) (1 + \delta), \quad |\delta| < \epsilon/2,$$

(حيث ϵ هي « epsilon - المكنة » ، أي الفرق بين 1 والعدد الأصغر المتحرك والمختلف عن 1) ، كما يمكن أن نستعمل (3.1) لتحليل الأخطاء . ولكن هذا لا يتم بشكل عام . هناك صعوبة أخرى [BROWN 77] ، هي المقارنات . فلنقم مثلاً بتخصيص $Comp := x > y$ ، حيث $comp$ هي منطقية (LOGIQUE) . فإذا كانت $comp$ بقيمة حقيقية (TRUE) ، نقول مع Brown « إن المكنة تُعلن إن $x > y$ » . والمشكلة هي أنها فعلاً تُعلن ذلك ، ولكن لا يوجد أي شيء يؤمن إن $x > y$ ، أي إن الأعداد الحقيقية المثلة بواسطة مضمون x و y هي فعلاً في هذا الترتيب . وقد يكون معنا $x < y$!

مفهوم Brown ، المتبوع من المؤلفين للغة آدا ، يقوم على الإقلاع عن وصف عدد للنوع FLOTTANT لمكنة معينة والنوع MODEL . هذه المُسلّمات هي « واقعية » ، في هذا المعنى ولكل مكنة ، يمكن تعريف أعداد نماذج بشكل نستطيع معه من التدقيق بها .

إنطلاقاً من هذه المُسلّمات ، أثبت Brown إن النظريات التي تسمح بإجراء إثباتات في بعض النقاط من البرنامج . مثلاً ، بعد التخصيص $Z := x * y$ ، من الممكن أن نؤكد أن :

$$(3.2) \quad |z - x * y| < \epsilon |x * y|$$

أو بعد

$Comp := x > y$ ، إذا كان $comp$ هو حقيقي ، وإن :

$$(3.3) \quad y < x(1 + \epsilon)$$

ولكن ϵ ليست هي ϵ - المكنة ، وهي مرتبطة بمجموع النماذج كما سنرى لاحقاً [MR 3.5.8] .

3.2 الأعداد النماذج (MODELE)

لنر بشكل دقيق ما هو النوع MODELE ، وما يناقضه في آدا .

3.2.1 تعريف وملاحظات

لنفترض أن t هو عدد صحيح أكبر من صفر . « الأعداد النماذج من الفئة t » هي صفر وجميع الأعداد الحقيقية بالشكل :

$$(3.4) \quad \pm 2^i (x_1 2^{-1} + \dots + x_t 2^{-t})$$

مع $x_1 = 1$ أو $x_1 = 0$ ، $2 \leq i \leq t-1$ ، ϵ ، صحيح ، موجود بين $4t-1$ و $4t-2$. (4)

في لغة آدا ، لا نذكر مباشرة t ، ولكننا نذكر عدد الأرقام العشرية ، وذلك بكتابة ، مثلاً :

$$(3.5) \quad \text{type T is digits k}$$

حيث k هو عدد صحيح أكبر أو يعادل 1 . نحسب إذاً الأرقام t في التمثيل الثنائي بواسطة :

$$(3.6) \quad t = \lfloor k \log 10 / \log 2 \rfloor$$

من الواضح ، إنه ليس من الضروري تثبيت الدقة التي نرغب بها في الأعداد الحقيقية بشكل واضح ؛ يوجد نوع FLOAT محدد سابقاً ، حيث k ، الذي يتعلق بالمكنة ، هو معطى بواسطة (FLOAT * DIGITS) .

فلنسم الآن فصحاح - نماذج (intervalles-modeles) الفصحاح الحقيقية ، حيث أطرافها هي أعداد نماذج ، أو $+\infty$ أو $-\infty$ (في هذه الحالة ، نقول أن الفصححة « تفيض ») . ستؤلف مجموعتها النوع MODELE (حسب إصطلاح جرى إدخاله للتبسيط ، وهو يختلف عن إصطلاح براون ، أو ذلك الخاص بـ [MR]) ، بعد أن تم تحديد عمليات وعلاقات عليه . وهذا النوع يحتوي على الأعداد النماذج نفسها ، لأنه من الممكن دائماً جمع العدد الحقيقي a في الفصححة $[a, a]$.

تعريف الفعل + ، - ، × (وكذلك بالنسبة للمؤثرات الموحدة / 1 -) على
الفسحات نماذج (1) ، تعتبر أولاً الفسحة الحقيقية التي نحصل عليها عند البحث في صور
أزواج الأعداد الحقيقية التي تنتمي إلى الفسحات العاملة ، وبعد ذلك يُوسَّع هذه الفسحة
(التي ليست فسحة نموذج) بزيادة فسحة نموذج صغيرة تحتويها .

البنية المعرفة على النماذج MODELES هي إذا جبر فسحات MOORE] Moore
[66 .

نعرف إذا علاقة بترتيب جزئي : إذا كانت x' و y' عبارة عن فسحات نماذج متصلة ،
سنشير إلى $y' > x'$ أو $y' < x'$ حسب الحالة . وإذا كان الطرف الأيمن لـ x' هو الطرف
الأيسر لـ y' ، سنشير إلى ذلك بـ $y' \leq x'$. وفي النهاية ، إذا كان x' و y' يتقاطعان في أكثر من
نقطة ، فالعلاقة ليست مغلقة .

فلنذهب الآن إلى العلاقات بين النوع FLOTTANT والنوع MODELE . إلى
كل عدد بفاصلة متحركة x نقوم بإجراء تناسب مع الفسحة الصفري النموذج x' التي
تحتويها . بينما القاعدة العامة لمسلمة براون هي : بدون تحديد دقيق لفعل العمليات على
أعداد متحركة ، نسلم بأنه متوافق مع نتيجة العملية على النماذج المضافة . وبشكل أكثر
دقة ، \mathbb{R} تعني واحدة من العمليات + ، - ، × ، المسلمات الأربع هي (مع إهمال
الدقة ، المعطاة من قبل براون ، على حالات الاسهاب :

$$1) \quad \Pi (x \oplus y) \in x' \oplus y'$$

$$2) \quad \Pi (x^{-1}) \in (x')^{-1}$$

$$3) \quad \Pi (-x) = - \Pi (x)$$

$$4) \quad \text{أ - إذا كان } y' > x' \text{ (أو أقل) ، فالممكن يجب أن تعلن إن } x > y \text{ (أو أصغر) .}$$

$$\text{ب - إذا كان } y' \geq x' \text{ ، فمن الممكن أن تعلن إن } x > y \text{ ، أو } x = y$$

$$\text{ج - إذا كان } y' = x' \text{ ، فمن الممكن أن تعلن أي شيء .}$$

(في 4 ، هناك تداخل في الحالة ، لأن x' و y' ، المضافة إلى x و y ، هي متساوية إذا
التقت في أكثر من نقطة) .

3.2.2 الخصائص

براون وصف مسلماته « بالواقعية » لأنه من الممكن دائماً ، وعلى مكنة معينة ،

(1) المبرور إلى العكس لا يُعمد إلا بالنسبة للفسحات التي لا تحتوي 0

تعريف مجموعة على الأقل من الأعداد النماذج (أي عدد صحيح t كما في 3.4) ، بشكل تتحقق فيه السُّلَّمات الأربعة . يكفي أن نأخذ t صغيراً .

إيجاد الأعداد t الملائمة لكثرة معينة ومصرف معين قد يكون صعباً ، ويفترض في أغلب الأحيان معرفة مفصلة للنظام . ولحسن الحظ ، هذا العمل ليس من واجب المبرمجين ، بل من واجب صانعي المصروفات . وفي هذا المجال ، يتطلب المعيار ADA إيجاد لكل من هذه الأنواع المسجلة $SHORT-FLOAT$ ، $LONG-FLOAT$ ، مجموعة من الأعداد نماذج والسماح للبرنامج ببلوغ ونيل t (أو k ، عدد الأرقام العشرية المناسب) بواسطة الخاصية $MANTISSA$ (أو $DIGITS$) . مثلاً ، لحاسب بثلاثة أرقام ثنائية ذات دلالة وبدون رقم آمان ، العدد $t = 3$ سيعرف مجموعة من الأعداد نماذج كبيرة جداً ، والتي لن تتحقق المسألة لها . وعلى العكس ، $t = 2$ هو أكثر ملائمة . أو بشكل عام ، $t = 2^{1-t}$ ، هو الفسحة القصوى بين عددين نماذج من فئة t ، أي البعد بين 1 والعدد نموذج الأعلى مباشرة . هنا ، هذا الـ «epsilon-modèle» يساوي $1/2$. وهو أكبر من «epsilon-machine» (الذي يساوي $1/4$) ، هذا الأمر هو عام .

عندما يقوم المبرمج بالتصريح عن النوع كما في (3.5) ، فهو يضيف إلى هذا النوع المجموعة $MODELES$ من فئة t ، حيث t معطاة بواسطة (3.6) (أو أكثر تحديداً مجموعة من الأعداد نماذج ، مفهوم النوع $MODEL$ الداخلي هنا لا ينتمي إلى لغة Ada) . يقوم المصنف بالبحث عن أي نوع $floatant$ معروف هو أكثر ملائمة ، أي الأقصر حيث المجموعة من الأعداد نماذج تحتوي على هذا المطلوب ، وهذا النوع يُؤخذ كنوع أساسي (إذا كان t المطلوب هو الأكبر ، والخطأ سيكتشف عند التصريف) .

منذ الآن ، بالامكان أن نركز على بعض الخصائص المتعلقة بالعمليات على الأعداد الحقيقية بفاصلة متحركة من نوع T . هذه الخصائص هي موجودة في النظريات (theorem) التي أوجدها براون من خلال سُلَّماته الأربعة .

النظرية 1 و 2 . إذا كانت الأعداد والتناجج عبارة عن أعداد - نماذج ، فنتيجة العمليات على أعداد بفاصلة متحركة هي دقيقة . النظرية 3 و 4 . إذا كان x و y هما عبارة عن أعداد نماذج ، إذا :

$$\Pi(x \oplus y) \leq (x \oplus y)(1 + \delta), \text{ et } \Pi(x^{-1}) = x^{-1}(1 + \delta), \text{ avec } |\delta| < \epsilon.$$

النظرية 6 - لنفترض x و y عبارة عن أعداد إيجابية . وإذا أعلنت المكتبة إن $x < y$ ، أي ما يعني $x < y(1 + \epsilon)$. وإذا أعلنت إن $x = y$ ، فهذا يعني إن $x < y(1 + 2\epsilon)$ و $y < x(1 + 2\epsilon)$.

النظرية 7 - لنفترض إن u عبارة عن عددتين غاذج ، x و y هي عبارة عن أعداد بفاصلة متحركة ، وجميعها إيجابي مع كون $x \leq u$ و $y \leq v$. فإذا $u \otimes v \leq x \otimes y$ هكذا ، وبإيجاز هي ، القواعد النظرية المعتمدة من قبل صانعي اللغة . فلن الآن ما هو الريح الذي يجنيه المبرمج .

3.3 نحو إختيار برامج رقمية

النقطة الأساسية هي إن ϵ موضوع التساؤل أعلاه ، ليست نفسها ϵ - المكنة . فهي تحت مراقبة المبرمج ، الذي يفرضها على المصروف عندما يكتب تصريجه (3.5) .

تعريف الأنواع بفاصلة متحركة في لغة آدا يؤمن التحقق من مسلمات براون . باستعمال نظريات براون ، يمكن للمبرمج أن يكتشف ، بواسطة الطرق الرياضية العادية ، خصائص متحولات في بعض النقاط من البرنامج (وأن يكتبها في البرنامج على شكل تأكيدات كما في (3.2) أو (3.3) - ومن المؤسف إن هذه التأكيدات لا يمكن أن تكون أكثر من ملاحظات ، في الصيغة الأخيرة لهذه اللغة) . هذه الخصائص تؤدي إلى تداخل Epsilon - نموذج ، وليس ϵ - المكنة ، أي تنتمي إلى البرنامج وإليه فقط . من الممكن أن ننصّر عمليات إختيار للبرامج مع الأخذ بالحسبان لظواهر الأخطاء في التنوير والبر (arrondir) . إضافة لذلك ، « فالخوارزم المكتوب بالارتكاز على الخصائص الدنيا التي يؤمنها التعريف عن النوع ، سيكون صالحاً للنقل بدون أي خوف » [p.3.13 MR] .

ومن الواضح ، أن صعوبة تحليل الأخطاء تبقى كاملة ، إضافة إلى صعوبة كتابة خوارزم (وأيضاً ضمناً ، اكتشاف صلاحيته) « بالارتكاز على ... » . وما تحمله لغة آدا ، هو إمكانية « الإرتكاز على » خصائص معروفة للبرنامج الذي سنكتبه وليس على خصائص المكنة أو المصروف التي نعرفها أو لا نعرفها ، الخ .

لا يجب أن نعتبر إن جميع مشاكل النقل من مكنة إلى مكنة والمرتبطة بالأخطاء الرقمية ستزول في لغة آدا . والبرامج ستواصل إعطاء نتائج مختلفة على مكنتات مختلفة . ولكن سيصبح من السهل التحقق ببعض المسائل المحلولة بشكل سيء حتى الآن : كيف نوقف خوارزم متكرر ؟ كيف نقوم باختيار وفحص دائرة ؟ الخ . أي التقدم نحو كتابة البرامج الصالحة للنقل من مكان إلى آخر أو من مكنة إلى أخرى . فلنلاحظ إن ϵ قد تكون أكبر من ϵ - المكنة : كي يتم إرضاء مسلمات براون ، ليس أمام المصروف عملياً سوى الاختيار بين نوعين أو ثلاثة أنواع مكددة سابقاً ، FLOAT ، LONG-FLOAT ، الخ . إذاً سنحصل في بعض الأحيان على أكثر من رقم ذي دلالة من المطلوب ، والنتائج

ستكون أكثر دقة مما تعلنه التأكيدات الموجودة في البرنامج . نقبل إذاً بخسارة قسم من المعلومات الموجودة في النتائج . وفي المقابل ما يبقى سيكون مستقلاً عن المكنة - الخاصة .

من الممكن دائماً ، بلوغ (بواسطة المعرفات «FLOAT MACHINE RADIX» و «FLOAT MACHINE-MANTISSA») القاعدة وعدد الأرقام من الجزء العشري ، المستعملة في الأعداد بفاصلة متحركة في الدقة البسيطة (أي بشكل غير مباشر ، على epsilon - المكنة) . ليس هناك ما يمنعنا من العودة إلى تحليل الأخطاء العادية ، إذا رغبنا بذلك .

كي نلاحظ بشكل دقيق ما تحمله لغة آدا ، يكفي أن نلاحظ أن مُسلمات براون تسمح لنا أيضاً بالحصول على إستعلامات عن كيفية تعامل البرامج بلغة فورتران مثلاً . فلنفترض مجموعة مختلفة من الآلات ، يوجد عدد t صغير جداً (أي نوع نموذج MODELE دقيق للغاية) كي يصبح متكيفاً مع الأنواع FLOTTANT لجميع المكنات (وهناك آخر للأعداد بدقة مزدوجة) . من الممكن أيضاً صياغة تأكيدات على هذه البرامج الصالحة على هذه الطبقة من المكنات . التقدم الذي حصل مع آدا هو في أنه يمكن تثبيت العدد t بواسطة المبرمج ، مع تأمين إن هذه التأكيدات المختبرة سيتم التحقق منها .

فلنشر ، لانتهاء هذا الموضوع ، إن هذا المفهوم جرى تطبيقه أيضاً على الأعداد الحقيقية بفاصلة ثابتة (الأعداد - نماذج في هذه الحالة ، هي عبارة عن أعداد مضاعفة لأعداد صحيحة بقيمة حقيقية ، تدعى «Delta») .

3.4 خاتمة

في حقل الأنواع الرقمية ، تستطيع لغة آدا أن تستعمل بشكل أفضل « هذا الفن » . مشاكل الأخطاء الناتجة عن التدوير والبت ستكون مفهومة بشكل أفضل ، كما سنجد أيضاً عند العمل بلغة آدا ، تشجيعاً للبحث عن إختبار صلاحية البرامج الرقمية .

الفصل الرابع

الأسماء والتعابير

4.1 مدخل

التعبير هو عبارة عن صيغة تعرّف عن طريقة حساب قيمة معينة [MR 4.4] ، ويتألف من متغيرات ، ومؤثرات وأشكال مراقبة وتدقيق and then or else .

نوع التعبير يتعلّق بشكل عام بالمؤثرات وينوع المتغيرات التي يحتويها : وقد يكون حسب النص ، بسبب التحميل الزائد أو المشكلة المطروحة بواسطة نوع المتغيرات .

4.2 المتغيرات

المتأثر قد يكون عبارة عن :

- متأثر حرفي literal

- مجموع

- إسم (خاصية أو إسم موضوع أو غرض)

- مُخصّص

- نداء لدالة

- تحويل للنوع

- تعبير يميز

- تعبير بين هلالين .

إضافة لذلك ، فبالنسبة للمؤثر in ، قد يكون المتأثر عبارة عن فسخة أو مؤشر عن نوع ثانوي .

4.2.1 المتغيرات الحرفية

يعني المتأثر الحرفي قيمة واضحة لنوع معين ، وقد يكون المتأثر الحرفي :

- رقمي : 3.14159-26536, 1-234

- ترقيم (رمزي أو سمات) : ROUGE 'A'

- سلسلة سمات : «TEXTE»

- الكلمة المحجوزة null ، التي هي عبارة عن قيمة ممكنة لأي نوع بلوغ ، والتي تسمح بترجمة كون الموضوع من نوع بلوغ لا يعني أي شيء .

4.2.2 المجاميع Agregats

المجموع agregat يعني قيمة من نوع جدول أو فقرة مع توضيح قيم مركباته .

تؤشر المركبات بشكل موقعي أو اسمي ، يجب أن تكون هذه المركبات محدّدة ؛ أمّا المركبات المحدّدة بشكل إسمي فيتم تحديدها بواسطة عملية إختيار بنحو شبيه بالأقسام المتحوّلة للفقرات ؛ الاختيار others ، الموضوع في النهاية ، يسمح بتعيين القيم غير المؤشّرة ؛ وإذا جرى استعمال الطريقتين ، يجب أن تظهر المركبات المحدّدة بشكل موقعي في البداية .

4.2.2.1 مجموع من نوع فقرة

فلنفترض ثلاثة تصريحات من الأنواع :

```
type DATE is
record
  JOUR : INTEGER range 1..31 ;
  MOIS : NOM_DE_MOIS ;
  ANNEE : INTEGER range 1000..2000 ;
end record ;

type COULEUR is (RIEN, ROUGE, BLEU, VERT) ;

type DESC is
record
  NO : INTEGER ;
  NBRE : INTEGER range 0..10_000 := 0 ;
  COUL : COULEUR := RIEN ;
  RECU : DATE ;
end record ;
```

- التعبير الموقعي :

```
( 14, JUILLET, 1789 )
( 789, 12, ROUGE, ( 7, DEC, 1980 ) )
```

- التعبير الأسمي (ترتيب مختلف)

```
( ANNEE => 1789, JOUR => 14, MOIS => JUILLET )
( NO => 789, RECU => ( 7, DEC, 1980 ), NBRE => 12,
  COUL => ROUGE )
```


- التعبير المختلط :

(14, ANNEE=>1789, MOIS=>JUILLET)
(789, 12, RECU=>(7, DEC, 1980), COUL=>ROUGE)

4.2.2.2 مواضيع من نوع جداول

لنفترض التصريحات :

type DOUZE is array (1..12) of INTEGER ;
D : DOUZE ;
M : array (1..3, 1..2) of INTEGER ;

- التعبير الموقعي

D := (7, 3, 4, 8, 5, 6, 1, 2, 11, 12, 10, 9);
M := ((1, 2), (3, 4), (5, 6));

- التعبير الاسمي (مركبات مؤشرة بواسطة دلائل) :

D := (1..12 => 0);
D := (1..3 => 1, 4..7 => 2, 8..12 => 0);
M := (1..3 => (1..2 => 0));
D := DOUZE' (1..3 => 1, 8..12 => 0, others => 2);
D := DOUZE' (others => 0);
D := DOUZE' (2|4|6 => 1, 3|5 => 2, others => 0);

- تعبير مختلط ، فقط الإسم others يمكن أن يظهر في القسم الاسمي .

D := DOUZE' (1, 5, 3, 7, others=>99);

التقدير

● هذه التعابير تسمح بتعيين أية قيمة من نوع جدول أو فقرة ؛ هكذا ، يمكن أن يكون نفس المجموع (agregat) من عدة أنواع ، ويحدد النوع الفعلي حسب الاستعمال .

● في بعض الحالات ، يجب تمييز المجموع (أنظر الأمثلة الثلاثة السابقة) ، وبالتحديد إذا كان جدولاً ، فهو يحتوي على الاختيار others ، والنص الكامل لا يسمح بتحديد كم هو عدد المركبات المعتمدة بواسطة others ؛ لن يكون بإمكاننا تبسيط عمل المصنف ، وعمل القارئ ، وذلك بجعل تمييز المجاميع إلزامياً ؛ ومن المحتمل أن يقوم المستعملون بإدخال هذا الأطناب بشكل مستمر .

● للمجاميع من نوع فقرة ، قد يؤدي من علم السماح إلا بتعبير واحد ، إلى تحسين إمكانية قراءة البرنامج ، كما هو الحال بالنسبة للنوع جدول . أما قواعد كتابة المجاميع (موقعية قبل الاسمية ، others في النهاية ، ...) فلا تظهر أبداً في النحو .

● وفي النهاية ، من المؤسف أن لا يكون النحو أكثر طبيعية : ..

(ANNEE := 1789 ; JOUR := 14 ; MOIS := JUILLET)

هذا التعبير سيكون أكثر ملائمة .

نفس الشيء ، الفاصلة ستحل محل الفضيض العامودي في عمليات الاختيار ، التي ليست هي عمليات اختيار للمواضيع أو الفقرات .

4.2.3 الأساء

يعني الاسم كثيراً من الأشياء في لغة آدا : فهو مُعرّف ، كما في بقية اللغات ، ولكنه عبارة أيضاً عن مرجع لعنصر أو عدة عناصر من متحولة ، نداء لدلالة ، أو سلسلة من السمات تُشكّل مؤثراً أو خاصية .

الأساء الوحيدة المسموحة كمتأثرات هي الخاصيات ذات القيمة وأسماء المواضيع (موضوع في كامله أو مركبات) .

4.2.3.1 attributes

- القيمة الصغرى للنوع COULEUR'FIRST- COULEUR

- عدد البتات لتمثيل قيمة من نوع DATE.SIZE . DATE

- الحد الأدنى للبعد الثاني للجدول TABLE'FIRST (2)

أسماء الخاصيات (FIRST, SIZE, ...) ليست محجوزة .

4.2.3.2 المواضيع (objects) أو الأغراض

الغرض أو الموضوع هو عبارة عن وحدة (متحولة ، ثابتة ، ...) ذات قيمة معينة . يُستعمل الترميز العادي للإشارات إلى مُرتّبات المواضيع الإنسانية :

. للفقرة

() للجدول

(...) للقطع

يستخدم الترميز المُقطّع للإشارة إلى كل وحدة غير مرئية مباشرة (انظر الفصل السابع) : وحدة من رزمة ، وحدة من فقرة ، مدخل عمل ، أو منهاج ثانوي .

يستخدم التلليل للإشارة إلى عنصر من مجموعة داخلية (انظر الفصل الثامن) .

التقدير

● على عكس باسكال ، فإنّ عمليات الترميز التالية :

- للتدليل والإشارة إلى جدول ببعدين A(I,J)

- للتدليل إلى جدول من جدول هي غير متعادلة A(DU)

● يُرمز إلى التأشير بواسطة () ، وليس بواسطة [] ، هذا الترميز هو أكثر قابلية للقراءة عند استعمال الترميز الآخر لأهداف أخرى . وهذا الاختيار يُوضَّح لسببين : صيغة المراجعة الموحدة الشكل وتحديد عدد السمات في المجموعة الأساسية .

● ليس ممكناً اختيار قطعة إلا من جدول يبعد واحد ، مما يشرح بصعوبة إنشاء جداول من خلال قطع وأجزاء من جداول الجداول .

● بلوغ الموضوع أو الغرض المؤشر بواسطة p ، نرسم إليه بواسطة p.all ، إذا أردنا أن نبليغ الموضوع بالكامل ؛ وعلى العكس ، عند مراجعة مُركَّباته ، يصبح الاختيار all غير مفيد بسبب عدم وجود أي إيهام ؛ هل هو ممنوع ؟ [MR] لا يسمح بالتجزئة والتقطيع ، حتى ولو اعتبرنا مسموحاً ما هو ممنوع بشكل جلي .

4.2.4 المخصصات allocateurs

يُشير المخصص إلى إنشاء الموضوع ، وذلك بالإشارة إلى نوع الموضوع المنشئ ، ومن المحتمل أيضاً أن يُشير إلى القيمة الأولية لإعداد الموضوع ، والتي يُشار إليها بواسطة مجموع المواضيع من نوع جدول أو فقرة . إضافة لذلك ، فالموضوع من نوع غير الزامي (جدول أو فقرة مع مميز) يجب أن يصرَّح عنه بتحديد الإلزام (الدليل أو المميز) .

يتعلَّق نوع المخصص نفسه بالنص :

- أمثلة على المخصصات :

```
type ENTIER is access INTEGER ;  
NOUVEAU : ENTIER ;  
NOUVEAU := new INTEGER ;
```

- إنشاء عند صحيح

```
type ELEMENT :
```

- تصريح غير كامل يصبح إلزامياً
بالرجعة إلى الأمام بواسطة LIEN

```
type LIEN is access ELEMENT ;
```

```
type ELEMENT is  
record  
  VALEUR : INTEGER ;  
  SUIVANT : LIEN ;  
end record ;
```

```
new ELEMENT ( 0 , null )
```

- إعداد اختياري يعادل

```
new ELEMENT ( VALEUR := 0 , SUIVANT := null )
```

```
type MATRICE is array ( INTEGER range < .. ,  
  INTEGER range < .. ) of REAL ;
```

type MATRICES is access MATRICE ;

new MATRICE (1..10, 1..20)

new MATRICE (1..10 =>(1..20 => 0.0))

- إلزام الدليل

- إعداد

واحدة من عمليات الإعداد السابقة هي إلزامية .

التقدير

يحتوي النحو على إبهام في قراءة [MR] ؛ وبشكل خاص .

new ident (expression)

يمكن أن تكون عملية تصفير لقيمة من نوع بسيط ، لقيمة من نوع مركب ، أو إلزام للمميز .

● قد يكون حرجاً تحديد نوع المخصص ، لأن الموضوع يمكن أن يُراجع بواسطة عدة قيم لنوع البلوغ ، أو لأنواع مختلفة .

قراءة [GI4.8] تعطي فكرة واضحة عن تعقيد المسألة : هكذا مخصص ، هل هو مسموح ؟ قد يدولنا غالباً إن هذا يشكّل صعوبة بالنسبة للمصنف .

● عدد من المسائل المعروضة بواسطة المخصص الديناميكي في لغات أخرى يجري تفادياً في لغة آدا ؛ هكذا ، فالتهديم الضمني للمواضيع يلغي إمكانية المراجعة المعلقة ، بينما عدم إمكانية تغيير الدلائل والمميزات تؤدي إلى أداء جيد في عمل الأواليه .

4.2.5 نداء الدوال

ABS هي الدالة الوحيدة المحلّدة بشكل سابق .

● لماذا لم يجري إختيار رمز لتمثيل القيمة المطلقة وإعطائها نفس التشريع كغيرها من المؤثرات ؟ مثلاً $abs\ X$ بدلاً من $ABS(X)$.

4.2.6 تحويل الأنواع

Type-conversion :: = type-mark (expression)

حيث type-mark يدل على النوع الذي سيجري تحويل التعبير expression إليه .

لغة Ada هي عبارة عن لغة متنوّعة ، وإختلاط الأنواع غير مسموح به إلا في بعض الحالات الشاذة جداً في التعابير ، لهذا فمن الضروري التصريح عن تحويلات الأنواع بشكل جليّ وواضح . هذه التحويلات هي ممكنة بين :

- الأنواع الرقمية .

- الجداول : المؤشرات والمركبات يجب أن تكون من نفس النوع أو من أنواع مشتقة .
- أنواع مباشرة أو غير مشتقة بشكل غير مباشر [MRA] .
- جميع التحويلات هي ممكنة في الاتجاهين : إذا كان في الإمكان تحويل قيمة من نوع معين إلى نوع آخر ، فالعكس هو ممكن ، وتستعمل هذه الصفة بالنسبة للمتغيرات الوسيطة الناتجة (out أو in out) للبرامج الثانوية .
- في حالة الجداول ، التي قد تحصل على حدود مختلفة في النوعين ، فحدود النتيجة ستكون من نوع الواصل ، إذا كان مشروطاً بإلزام ، وإلا فهي من نوع التعبير المطلوب تحويله .
- أمثلة على تحويلات الجداول .

```
type SEQUENCE is array (INTEGER range <>) of INTEGER ;
subtype DOUZE is SEQUENCE (1..12) ;
SUITE : array (1..100) of INTEGER ;
```

- حدود = تلك الخاصة بالنوع .
- إغفال SUITE
- الحدود = 31 و 42
- الحدود هي تلك الخاصة بـ DOUZE

• conversions de types dérivés

```
type A is new B ;
X : A; Y : B;

X := A (Y) ;
Y := B (X) ;
```

التقدير

- Ident (expression) هو عبارة عن شغل نحوي يستخدم في كثير من الأعمال .
- لحسن الحظ لا تستطيع أدا القيام بأعمال تحويل ضمنية ، لأن هذا قد يجعل من غير الممكن اختيار الوحدات المعلقة للشحن .
- تحديد حدود نتيجة تحويل الجداول هي أكثر تعقيداً .

4.2.7 تعابير مميزة

- وصف (أي التأثير الواضح عن النوع) التعبير سيكون ضرورياً بواسطة الإمكانات التي تقدمها لغة آدا ، أو عند تحديد نوع المواضيع أو الأغراض .
- المسألة ستفرض نفسها بشكل خاص للترقيمات ، التي قد تحصل على قيم من أنواع مختلفة :

مثلاً :

type MASQUE is (FIX, DEC, EXP, SIGNIF) ;
type CODE is (FIX, CLA, DEC, TNZ, SUB) ;

MASQUE'(DEC)

CODE'(DEC)

DOUZE'(1131517 => 2, others => 0)

- الملائين () هما الإزمانيان .

- من نوع CODE

من الممكن أن نجد بعض الملاحظات على هذا الموضوع في [Moffat'8] .

4.3 المؤثرات

هي حسب ترتيب الأولوية المتصاعدة :

منطقية and or xor

علاقات = / < > =

للجمع والطرح + -

توحيدية + - not

للضرب والقسمة * / rem mod

%01gge.n

فقط هذه المؤثرات يمكن أن تكون مشحونة ومحددة ، ما عدا التعادل = د = / التي لا يمكن أن تحدد إلا بالنسبة للأنواع الخاصة (انظر الفصل 7) .

التقدير

● إمكانية تحميل المؤشرات هي شديدة الأهمية ، مع إنها قد تكون مُملة عند القراءة ؛ ومن الممكن تعريف ، واستعمال ، بشكل سهل ، مثلاً العمليات على المصفوفات أو أيضاً العلاقات بين الأنواع المختلفة .

● قد نأسف لعدم إمكانية تصريح رموز جديدة للمؤثرات ، مع إن هذه السهولة ستؤدي إلى إدخال أوالية جديدة في تحديد الأولويات .

4.3.1 - المؤثرات المنطقية

نوع النتيجة	نوع المتأثرات
BOOLEAN	BOOLEAN
نفس نوع المتأثرات	جدول بالتحويلات البولية يحدد واحد وله نفس عدد المركبات

تقدير المتأثر الموجود لجهة يمين and or سيتم تقديره بشكل واضح وليس باستعمال أشكال المراقبة المختصرة (دائرة قصيرة) .

A and then B _

- لن يتم تقدير B إلا إذا كانت A حقيقة (True) .

A or else B

- لن يتم تقدير B إلا إذا كانت A خطأ (false) .

أشكال المراقبة هذه تتمتع بنفس الأفضلية التي تمتاز بها المؤثرات المنطقية ، وهي لا تمثل مؤشرات .

من غير الممكن خلط عدة مؤثرات منطقية ، أو أشكال موجزة ، بدون أهلية :

A and B and C

- صحيح

A and B or C

- غير صحيح

(A and B) or C

- صحيح

N = 0 or else A(N) = 1

- صحيح

A and B and then C

- غير صحيح ، يجب أن يكتب :

(A and B) and C

4.3.2 علاقات وانتهاءات

نوع النتيجة	نوع المتأثرات	مؤثرات
BOOLEAN	أي نوع	= /=
BOOLEAN	نوع ساكن لا إجهادي أو جداول يبعد واحد ويمركيات مُفردة	< <= > >=

كما بالنسبة للمؤثرات المنطقية ، لا يمكن مزج عدة علاقات بدون أهلية :

A = B = C

- غير صحيح

(A = B) and (A = C)

- صحيح

تجري مؤثرات الانتباه (in) وعدم الانتباه (not in) على متأثرات من الأنواع التالية :

المتأثر اليميني	المتأثر لجهة اليسار	النتيجة
فسحة		BOOLEAN
مؤشر النوع - الثانوي	نوع حدود الفسحة النوع الأساسي للنوع - الثانوي	BOOLEAN

أمثلة :

N not in 1 . . 10

AUJOURDHUI in JOURS range LUNDI . . VENDREDI

التقدير

● يدو in not ليست عبارة عن مؤثرات (لا يمكن تحميلها كثيراً) ؛ ولكننا لا نرى جيداً ما يمكن أن نستطيع القيام به ؛ ميزتها الوحيدة هي إن المتأثر اليميني لهذه العلاقة هو بدون قيمة .

● in هي شبيهة بتلك الموجودة في لغة باسكال ، ولكن لن يكون باستطاعتنا تطبيقها على مجموعة قيم غير منظّمة ، بدون تعريف النوع - الثانوي ؛ وعلى العكس لن يكون بإمكاننا تطبيقها على سلسلة من القيم غير المتواصلة أو المتجزئة .

● لا يوجد مجموعات في لغة آدا ، ولكن المؤثرات المنطقية والعلاقات هي معرفة على جداول متحولات منطقية : وهذا هو الوضع أصلاً .

4.3.3 مؤثرات جبرية

ما يتبع لا يتعلّق إلا بالمؤثرات المحددة سابقاً ، فلتذكر بأنه يمكن إعادة تعريفها .

+ - موجدة

+ - ثنائية ، and (إتمام)

rem, mod / *

**

الإلتزام ينطبق على متأثرين من نفس نوع جدول بعد واحد ، أو من نوع مركبات . المفهوم العام يحدّد بشكل عام نوع النتيجة [MRA] .

ما عدا في الحالات الشاذة الاستثنائية [MR 4.5.5] ، الأنواع الرقمية لا يمكن خلطها ، والعلاقات التالية مثلاً :

- هي ممنوعة إذا كانت A + 0.1

ليست - من نوع حقيقي

- هي مسموحة إذا كانت A عبارة عن $A * 2$ عدد صحيح أو حقيقي ثابت .

التقدير

من الأولويات المحلدة معنا :

$$\begin{aligned} -A + B &=> (-A) + B \\ -A * B &=> -(A * B) \\ -A \circ B &=> -(A \circ B) \end{aligned}$$

● 'A' and 'B' يبدو وكأنه ممنوع بواسطة [MR] . ولكن هذا مسموح به بشكل جليّ [MRA 4.5.3] . ولكن ماذا يجري إذا كان النص لا يسمح بتعريف كامل لنوع النتيجة .

4.4 تعابير ساكنة ، تعابير حرفية

نحسب التعابير الساكنة بواسطة المصروف ، وهي لا تستطيع أن تحتوي على قيم محسوبة بشكل ديناميكي . التعابير الحرفية هي عبارة عن تعابير ساكنة خاصة محلدة بقيم صحيحة عامة وحقيقية عامة . وتستعمل لاعطاء قيم للاعداد عند التصريح عنها ، وبعض القيود هي مفروضة على المتأثرات والمؤثرات .

التقدير

● [MRA] حلد الفرق بين التعبير الساكن والتعبير الحرفي .

● معالجة الشواذات التي تظهر في التعابير الساكنة ليست واضحة: هل تسمح للريعة بترحيلها إلى التنفيذ ، أو بإهمالها ؟ حسب [MR 1.6(2)] يستطيع المصروف ، في الحالة المثل ، تحديد إن الخطأ سيتم إكتشافه عند التنفيذ .

خاتمة

بالنسبة لما يتعلق بالمؤثرات ، فلهذا آدا هي أفضل من اللغات الأخرى ، مع إن النحور ونظام الأولية هو غير عادي (يختلف عن باسكال تحديداً) ؛ النقطة الأكثر إيجابية هي في إمكانية إعادة تعريف هذه المؤثرات . بالنسبة للمتأثرات ، تظهر بعض المشاكل ، ناتجة عن التعقيد في التدقيق بتكثيف الأنواع (وبالتحديد نسبة إلى الأشياء ، أكانت متأثرات أو قيم إعداد للمواضيع الديناميكية) ؛ وفي النهاية ، النحول ليس دائماً سهلاً ، وإمكانية قراءة البرنامج تصبح صعبة .

الفصل الخامس

تركيبات المراقبة المتتالية

هذا الفصل يعالج القسم الأكثر كلاسيكية في لغات البرمجة ، أو القسم الذي يشرح التعليمات ومسائل مراقبة تنفيذها : تدقيق ، تكرار ، تفريع ، الخ [MR5] . في أغلب الحالات ، التعليمات المستعملة فقط في الأعمال التعاونية ستعالج في الفصل المناسب (الفصل 8) ؛ نفس الشيء ، فتلک المشتقة من الإجراءات سيجري النظر إليها حسب مفهومها (الفصل 6) ، بما فيه التعليمات return ، المعرفة أيضاً في [MR5] .

هكذا ، فالتعليمات ، في لغة آدا ، تتبع العادة وذلك بالسماح ، كبرقية لغات البرمجة الجامعية الشائعة ، بما يلي :

- تعطي المستعمل إمكانية إختيار كاف للتركيبات للتحكم « بشكل طبيعي » بتنفيذ لائحة التعليمات .

- المحافظة على روح البرمجة التركيبية : لا يوجد طفوف إلى أمكنة مختلفة ، والتكيف مع المجموعات المنطقية .

- تحسين إمكانية قراءة البرامج (مثلاً : التعليمة Case) ؛ نرجع إلى الفصل 14 حول هذا الموضوع .

سنحاول هنا تحديد بعض النقاط ، وإعطاء بعض الأمثلة وإجراء بعض الانتقادات على هذه التركيبية .

5.1 التعليمات : تسلسل وتركيب البلوك

5.1.1 تقديم

لا تظهر التعليمة إلا في لائحة من التعليمات ؛ يُعلم التوالي بواسطة نقطة فاصلة تُهي كل تعليمة (أنظر الفصل 14 للمفهوم النحوي) .

يمكن تجميع سلسلة التعليمات في فقرة bloc تحتوي بدورها على قسم تصريحي . هذا يسمح بالتصريح عن الجداول الديناميكية في معنى (Algol 68) (حدود محسوبة في فقرة

مُغلّقة) ؛ إضافة لذلك فهذه الفدرات يمكن أن تستعمل ، في لغة آدا ، لمعالجة الاستثناءات .

وفي لغة آدا ، لا يوجد (كما في لغة Algol 68) أية إمكانية لتحديد إستقلالية التعليمات الموجودة دوماً بشكل متالي بداخل جسم معين (body) .

5.1.2 تقدير

على عكس لغة باسكال ، ولكن كما بالنسبة للغة Algol 60 ، تسمح الفدرات بكتابة جداول ديناميكية خارج الإجراءات ، مما يعني تفادي تكوين الإجراء الذي ، وبدونها ، سيكون غير مفيد . وغالباً لا يوجد طريقة للتحكم بطول هذه الفدرات (أو الإجراءات) ما عدا كون البرمجة التركيبية تتطلبها صغيرة .

5.2 التخصيص

5.2.1 تقدير

أولوية الحساب للقسامين ، الأيمن والأيسر ، ليس الزامية وغير معروفة في اللغة . ولا يوجد تحويل للنوع المعروف ضمناً ، كذلك بالنسبة للأنواع الرقمية (أنظر الفصل 2) .

لا يوجد إمكانية تخصيص متعددة . مثلاً ، التعابير التالية :

— ممنوعة : $A := B := C$;

— ممنوعة : $A, B := 0$;

— بينيا

$A, B : \text{INTEGER range } 1 \dots 10 = 0$;

هي مسموحة

وعلى العكس ، فمن الممكن تخصيص أجزاء بنفس الطول .

5.2.2 مثلاً :

بعد التصريحات

$A : \text{STRING } (1 \dots 31)$;

$B : \text{array } (1 \dots 31) \text{ of CHARACTER}$;

$C : \text{STRING } (3 \dots 33)$;

التعليمات التالية هي مسموحة :

$A := C$;

$A(1..5) := C(25 \dots 29)$

— جداول من نفس النوع ونفس الطول

— أجزاء من نفس نوع STRING

.. السلاسل يمكن أن تُخصّص إلى المتحولات : «Sauce béarnaise» : A (1.. 15)
 STRING وجداول السمات ذات البعد الواحد : «sauce bearnaise» : B (1...15)

من الممكن تخصيص وبشكل كامل متحولات مركّبة (فقرات ، جداول) من نفس النوع . في المثل السابق ، لا يوجد تخصيص كامل بين A و B وهما من نوعين مختلفين ، ولكن عملية التخصيص لكل عنصر مع عنصر آخر هي ممكنة .

5.2.3 تقديم

الصعوبة الوحيدة في التخصيص ، تظهر غالباً في تحديد ، وحسب نوع القسم الأيسر ، النوع المؤكد للتعبير الواقع لجهة اليمين في حالة العمليات على أقسام من الجداول ، أو على جداول بحدود قسرية إلزامية .

قد نأسف لعدم وجود تخصيص غتلط مع مؤثر جبري يسمح ، كما في Algol 68 ، بكتابات واضحة وإهمال تأثيرات الخواف ؛ مثلاً :

$$A(1 + F(J)) + : = 1 ;$$

بدلاً من :

$$A(1 + F(J)) := A(1 + F(J)) + 1$$

5.3 التعليمة if

5.3.1 تقديم

كل ما هو كلاسيكي غالباً . فلنذكر الأهله if ... end if وفعل وجود نقطة - فاصلة أمام else وأخرى أمام end if

```
if (COND) then
  INST1 ;
else if (COND)2 then
  INST2 ;
  else if (COND)3 then
    INST3 ;
    else INST4 ;
    end if ;
  end if ;
end if ;
```

يوجد شكل مكثّف شبيه بـ Algol 68 ، التعليمة السابقة يمكن أن تكتب كما يلي :

```

if COND1 then INST1 ;
elseif COND2 then INST2 ;
elseif COND3 then INST3 ;
else INST4 ;
end if ;

```

هذا الشكل المكثف للتعليمية if يسمح بتخفيض عدد if المتدرج وعدد الكلمات end if المناسب التي ستصبح ضرورية في الحالة الأخرى .
فلنشير إلى وجود دوائر مقطوعة (أنظر 4.3.1) تسمح بتكوين تسلسل الوصلات .

```

If DELTA > 0 and RAC2 (DELTA) < 3 then ... end if .

```

.. شواذ ممكن
هذا هو مغلوط ، لأنه ومهما يكن التعبير الأول أو الثاني يمكن أن يحسب أولاً لأن and وتفترض أن يكون متأثرهما مستقلين .

```

If DELTA > 0 and then RAC2 (DELTA) < 3 then ... end if ;

```

هذا هو صحيح لأن تقدير الشرط الثاني لا يتم إلا إذا كانت $DELTA \geq 0$ (التسلسل في تكوين متأثرات and then) . أنظر 4.3.1 بخصوص and then و or else ، اللذان يختلفان عن كونها متأثرات .

5.3.2 التقدير

● التعبير If...end if ، المعروف على الأقل منذ سنة 1968 ، هو حتى الأفضل لمقارنته مع باسكال مثلاً .

● إمكانية قراءة الدارات المقطوعة ، يمكن أن تصبح موضع شك في تعليمات مثل :

```

If A or else B then ... else ... end if ;

```

حتى ولو كُتبت على عدة أسطر .

● بشكل عام فإن الفقرة تحسن إمكانية القراءة .

5.4 التعليمية

5.4.1 تقديم

هي على الشكل التالي :

```

case EXP is
  when CHOIX_1 => ... ;
  ...
  when CHOIX_1 | CHOIX_2 => ... ;
  ...
  when others => ... ;
end case ;

```

كل قيمة للنوع أو النوع الثانوي من التعبير يجب أن تمثل مرة واحدة فقط في مجموعة الاختيارات ، الاختيار others يمثل جميع الاختيارات غير المذكورة . قيمة الاختيارات المختلفة يجب أن تكون قابلة للتحديد قبل عملية التصريف .
فلنشير إلى إمكانية استعمال تعابير مميزة لخصر عدد الاختيارات التي تحتاج فعلاً إلى التحديد .

5.4.2 التقدير

- هذا الشكل case هو شكل تركيبى بشكل جيد مثل الشكل LIS ، حتى لو اختلفت عن الأشكال المعروفة .
- نأسف لأن تصادف الكلمة others ليس نحويّاً (أنظر الفصل 14) .
- هذا النموذج هو مريح لتجميع الاختيار .

مثلاً :

VINTAGE : INTEGER range 1970..1980 ;

```

case VINTAGE is
  when 1970|1975 => ... ;
  when 1971..1974|1978 => ... ;
  when others => ... ;
end case ;

```

فلنذكر استعمال ...

- حيث الاختيار when هو أفضل من if أو «كما في Algol. 68»
عملية إختيار الكلمات - مفاتيح ليست دائماً موقّعة .
لماذا لن يكون then أو : بدلاً من => ?
لماذا لا تكون else أو others بدلاً عن when others
IS ليس لها أية فائدة نحوية بسبب وجود when .
- توضيح الأفعال بداخل أهلة خاصة ، بواسطة => when (أو end case) ، مما يعني نقطة جيدة .

5.5 الحلقات

5.5.1 التقديم

يوجد شكل أساسي يسمح بكتابة جميع الأشكال الأخرى (Basic loop) ، وغالباً بشكل إصطناعي . ويمكن أن تكون الحلقة الرئيسية مسبقة بشرط للتكرار ، يعطي الشكلين التاليين للحلقة .

```
while CONDITION loop
  INSTRUCTIONS ;
end loop ;
```

```
for VAR in [ reverse ] BORNES loop
  INSTRUCTIONS ;
end loop ;
```

الحلقة for تشبه الحلقة الموجودة في Algol 68 : تصريح مركزي وضمني عن متحولة التحكم (loop-parameter) إذاً غير القابلة للتغيير بواسطة المبرمج والمحدد للحلقة . فلنشير إلى إن متحولة التحكم يمكن أن تأخذ قيماً مختلفة من فسخة من الأعداد ، أي ما نستطيع كتابته :

```
for I in COULEUR loop ...
```

وفي الحلقات المتداخلة ، تؤدي التعليمة exit إلى الخروج من الحلقة الداخلية ، أو من الحلقة ذات الإسم المحدد .

```
while ... loop
  BOUCLE :
    while ... loop
      while ... loop ...
        exit BOUCLE when CONDITION ;
        -- If CONDITION then goto SUITE end If ;
      end loop ;
    end loop BOUCLE ;
  <<SUITE>>
end loop ;
```

إسم الحلقة يمكن أن يستعمل في تعبير يميز لتحديد المتغير الوسيط للحلقة :

في هذه الحلقة ، مؤشر الحلقة I ستراجع بواسطة BOUCLE .

```
procedure TIRER_AU_HASARD ( RESULTAT : out INTEGER) is
  ...
  BOUCLE : for I in 1..100 loop
    ...
```

```

declare
  BUFFER : array (1..1) of INTEGER :=(BUFFER_RANGE => 0);
  I : INTEGER;
begin
  ...;
  TIRER_AU_HASARD(I);
  -- mais dans ce bloc, le I de la boucle doit être référencé BOUCLE.I
  BUFFER ((BOUCLE.I + 1)/2) := I;
end;
end loop BOUCLE;
end TIRER_AU_HASARD;

```

5.52 التقدير

● غياب التصريح عن متحولة الحلقة هو مربع بالنسبة للمبرمج (الذي لن يهتم بها أو يخلطها مع دليل آخر) والذي المحدث هو أمر جيد لتفادي منبع الأخطاء وفقدان إمكانية القراءة .

● اللغة لا تعطي أية إمكانية لتعريف الحلقات المتوازية المستعملة بواسطة المكتبات

```

for all I in 1 .. 256 do
  A(I) := A(I) + B(I);
end do

```

- عرض

● لا يوجد حلقة مع إختيار في نهاية الحلقة كما في باسكال .

```

loop
  INSTRUCTION;
end loop until EXPRESSION;

```

- هذا الشكل
- لا يوجد
- في أدا

يبدو كأنه باستطاعتنا أن نعبّر عنها (بواسطة EXIT WHEN EXPRESSION) طالما إنه من الصعب أن نجد في أدا ، نحواً مربعاً ومتجانساً مع الحلقات الأخرى .

● غياب « الخطوة » هو مؤسف ، وبالأخص في التحليل الرقمي . المتغير المعطى في [ME P. 3.10] يبدو لنا الأقل مكرراً (هذه الأولوية ، الموجودة في اللغات الأخرى ، لن تكون مستعملة) . نفس الشيء ، نأسف لغياب « اللائحة التي » تسمح بتسهيل العمل على منحرفات المصفوفات (diagonale) ، مثلاً :

```

for J in 1..I-1 | I+1..N loop

```

هي غير موجودة في لغة أدا .

● وللأسف لا يوجد إمكانية (ما عدا بواسطة goto) للذهاب إلى نهاية الحلقة وبعد ذلك بإعادة التكرار كما في Jovial .

BOUCLE : for I in ... loop :

...
continue BOUCLE when CONDITION ; -- غير موجود

...
end loop BOUCLE ;

● النحو هو متجانس (يوجه دائماً) حلقة أساسية « تسمح بالقيام بكل شيء » ، وتؤمن عملية التجميع بداخل الألة بشكل بسيط . ولكن قد يخشى من أن لا يؤدي استعمال exist إلى نفس المخارج كما يؤدي goto . ونجد من المؤسف أن يكون إسم رأس الحلقة يشبه كثيراً الوسوم في بقية اللغات . لهذا لم يتم اختيار بعض الأشياء كما :

loop <<MA_BOUCLE>> - افتراض

A := A + 1 ;
exit MA_BOUCLE when A > 0 ;

...
end loop MA_BOUCLE ;

5.6 الطفور والوسوم

5.6.1 تقديم

في هذه اللغة هناك إمكانيات للطفور ، ولكن باستعمال جداً محدود : ولا يمكننا الدخول بواسطة goto إلى تعليمة مركبة ، ولا الخروج من إجراء ، الخ . عمليات الطفور ليست ممكنة وبشكل نهائي إلا عندما لا يكون هناك أية مشكلة في المحيط (الطفور يجب دائماً أن يعالج بواسطة jump وتعليمات آلية) .

يصرّح عن الوسوم ضمناً ، كما في فورتران أو في Algol 60 ! فلنشر إلى إن الوسوم end للتعليمة المركبة يجب أن يتم بطريقة غير مباشرة بواسطة تعليمة فراغ :

< < FIN > > null ; end

5.6.2 التقدير

يوجد منذ عدة سنوات ، ميل لإلغاء الطفور في لغة البرجة لأن هذه العملية تعتبر خطيرة جداً ، وتجعل البرامج أقل إمكانية في القراءة وتذهب إلى مواجهة طرق البرجة المتصاعدة . وفي لغة آدا ، تبدو عمليات الطفور هذه غير مفيدة لأن تعليمات التحكم هي معدة من أجل ذلك ، بدون شك فإن واضعي (دفتر الشروط) فضلوا ترك هذه التعليمة منجزة وبدون خطر بدلاً من الإجابة على انتقادات حرجة ؟ في أغلب الأحيان تسمح عمليات الطفور بمعالجة البرامج المولدة أوتوماتيكياً وقد يبدو مركزياً استعمالها عملياً .

مثلاً :

< < ETIQUETTE > > ..

...

goto ETIQUETTE;

5.7 تركيبات أخرى للتحكم المتتالي

لنذكر بأن التعليمة return سنراها في مفهوم الاجراءات (الفصل 6) وفي الفصل المخصص للأعمال .

بالإمكان برجة الأوتوماتون وجداول البرجة بواسطة تعليمات من نوع case ، مما لا يعكس بشكل صحيح الجداول المُرَبَّعة . لا يبدو هنا إن آذا قد حملت شيئاً جديداً .

5.8 خاتمة

النحو ينتج عن جهد في التنقية : جميع تركيبات المراقبة مجموعة بداخل أهلة ، ويمتاز بأسماء (nom) توضع في البداية أو النهاية .

الفصل السادس

التقسيم إلى زجل

يتألف كل برنامج آدا من مجموعة من وحدات برمجية يمكن أن تُصوَّف بشكل منفصل أو غير منفصل (أنظر الفصل 11) . وحدات البرمجة تناسب تقطيعاً منطقياً للبرنامج . سندرس في هذا الفصل البرامج الثانوية والرزم . الأعمال والوحدات النوعية ستدرس في الفصل 8 و 10 .

6.1 البرامج الثانوية

6.1.1 المواضيع المرتبة

البرامج الثانوية في لغة آدا تناسب المفهوم العادي للبرامج - الثانوية ، ولكن مع تحديد دقيق بين الدوال والاجراءات .

- الدوال تعيد قيمة من نوع معين وتمتاز بمتغيرات غير قابلة للتعديل .
- الاجراءات يمكن أن تعدل متغيراتها ولكنها لا تعيد أية نتيجة إلى نقطة النداء .

6.1.2 الوصف

البرامج الثانوية ، ككل وحدة برمجية ، تتألف من قسمين :

- المواصفة التي تناسب رأس البرنامج - الثانوي .
- جسم البرنامج الثانوي .

هذان القسمان يمكن أن يتباعا أو أن يكونا منفصلين (مثلاً ، المواصفة في القسم المرمي من الرزمة والجسم في الرزمة) . عندما يكون القسمان المنفصلين ، يجب أن تتكرر المواصفة أمام جسم البرنامج الثانوي . هذا الإلزام يسمح برفع الإبهام المحتمل عندما يكون هناك تحميل زائد . ولكن ذلك يزيد في صعوبة كتابة الجسم ، ويحسن من إمكانية القراءة .

البرامج الثانوية يمكن أن تكون محمّدة بشكل متكرر .

ويحتوي القسم الإلزامي على لائحة بالتصريحات عن المتغيرات الشكلية مع طريقة عبورها (أو استعمالها) :

- المتغيرات الوسيطة عبارة عن معطيات (الصيغة in) ويمكن أن تحصل على قيم بالغلط .
- متغيرات وسيطة عبارة عن نتائج (الصيغة out) .
- متغيرات عبارة عن معطيات ونتائج (الصيغة in out) .

نداءات البرامج الثانوية تبقى كلاسيكية ، ولكن التناسب بين المتغيرات الوسيطة الفعلية والإلزامية يمكن أن يتم إما حسب موقع المتغير (بطريقة موقعية) ، وإما بتسمية المتغير الشكلي . وبالإمكان إهمال متغيرات شكلية للصيغة IN عندما تكون المتغيرات الشكلية المناسبة مصرحاً عنها بواسطة قيم بالغلط .

العودة إلى الوحدة المنادية ، في الإجراء ، يمكن أن تتم إما أوتوماتيكياً بعد تنفيذ الإجراء ، وإما بواسطة التعليمة return . في حالة الدوال function ، تكون التعليمة return إلزامية ومتبوعة من التعبير - نتيجة .

البرامج الثانوية يمكن أن تكون محمّلة (أنظر 7.2) ؛ التحميل الزائد للمؤثر يتم بوضع إسم المؤثر بين مزدوجين (") .

6.1.3 تمرين 1

- أنظر المثال المستوحى من [MR 7.3] .
- سيتم مراجعته وتكملته في نهاية هذا الفصل .

```

type RATIONNEL is
  record
    NUM, DEN : INTEGER ;
  end record ;
procedure MEME: DENOMINATEUR (X,Y : in out RATIONNEL) is
begin
  X.NUM := X.NUM * Y.DEN ;
  Y.NUM := Y.NUM * X.DEN ;
  X.DEN := X.DEN * Y.DEN ;
  Y.DEN := X.DEN ;
end MEME: DENOMINATEUR ;
function EQUAL (X,Y : RATIONNEL) return BOOLEAN is
  U,V : RATIONNEL ;
begin
  U := X ;
  V := Y ;
  MEME: DENOMINATEUR (U,V) ;

```

- دعوة الإجراء السابق .

```
return U.NUM = V.NUM ;
```

- نتيجة الزامية للدالة

```
end EQUAL ;
```

قد يحصل أن يكون هناك في نفس وحدة البرمجة نوع آخر TOTO وبالتالي يجب أن يتم تطبيق EQUAL عليه .

```
type TOTO is
```

```
record
```

```
...
end record ;
```

```
function EQUAL (X,Y : TOTO) -- تحميل الدالة EQUAL
```

```
return BOOLEAN is
```

```
...

```

```
end EQUAL ;
```

```
...
declare
```

```
...

```

```
A,B : RATIONNEL ;
```

```
C : BOOLEAN ;
```

```
begin
```

```
C := EQUAL (A,B) ; -- نداء الدالة EQUAL
```

```
... -- المناسبة للأعداد الجذرية
```

```
end ;
```

6.1.4 المشاكل المختلفة

أ - غياب المتغيرات الإجرائية

لا يمكن أن تكون البرامج الثانوية نفسها عبارة عن مُتغيرات . وتستبدل هذه الأولية بأخرى أكثر شمولية حيث الاستعمال يبدو صعباً في بعض الحالات (أنظر المثل عن تكامل الدالة في 10.3.1) .

ب - موقع البرامج الثانوية في الأقسام الوصفية

جميع الأقسام الوصفية تتألف من مجموعتين وقد يُحتمل أن نكوناً فارغتين . المجموعة الأولى تحتوي :

- كلمات - مفاتيح :
use

- تصريح عن الثوابت ، المتحولات والأعداد

- تصريح عن الأنواع والأنواع الثانوية .

- تصريح عن الإستثناءات .

- إعادة تسمية .

ج - الفصل بين المواصفة وجسم البرنامج الثانوي
 الفصل بين المواصفة وجسم البرنامج الثانوي يفترض بعض الإحترازاات في الاستعمال : « لا يجب أن يُطلب البرنامج الثانوي خلال صناعة القسم الوصفي إذا كان جسم هذا البرنامج الثانوي موجوداً بعد الطلب أو بعد تعليمة التداء » [MR 3.9] .
 مثل رقم 3

```

...
function F ( . . . ) return INTEGER ;
I : INTEGER := F ( . . . ) ;    - خطأ ، إذا كان جسم F غير مُكوّن
...
function F ( . . . ) return INTEGER is ... end F ;
    
```

نجد نفس المشكلة في حالة الرزم .

هـ - مدى المتغيرات الشكلية
 مدى المتغيرات الشكلية هو نفسه كمدى البرنامج الثانوي حيث يتم التصريح عنها . ولكنها ليست منظورة من خارج جسم البرنامج الثانوي ، إلا عند دعوة هذا الأخير، وكي نتمكن من إقامة التناسب بين المتغيرات .
 مثل رقم 4

```

...
procedure P ( A : in INTEGER ; . . . ) is
...
end P ;
P ( A => I , . . . ) ;
    
```

الرمز (*) يدل على إمكانية رؤية A ، الرمز « | » هو مدى A .

و - عبور المتغيرات
 الطريقة التي يتم فيها عبور المتغيرات ليست دائماً مفروضة من صيغتها . مهما تكن هذه الطريقة ، فالمتغيرات بالصيغة in لا يمكن أن تتعدّل في البرنامج الثانوي .
 - هناك عدة مشاكل تفرض نفسها بالنسبة للمصنّف الأخرى .
 - إذا انتهى البرنامج بشكل غير اعتيادي ، فمضمون المتغيرات ذات الصيغة out و out in ليس ذا أهمية .
 للمصنّف الذي يختار العبور بواسطة عنوان ، فالصنّف out و out in هي متعادلة . لا

يمكن أن تتحكمم إلا برنامج ثانوي لا يستعمل متغيراً بسيطاً بصيغة out في القسم اليمين
للتخصيص قبل إجراء تخصيصه .

فقدان التوازن بين الصيغ in و out هو مؤسف .

إضافة لذلك فعند دعوة برنامج - ثانوي ، لا يوجد أي تحديد على المتغيرات .
الصيغة الحالية للغة هي أقل طموحاً من [GREEN] تلك التي تبحث عن إلغاء جميع
حالات الاستعارة القادرة على إحداث تأثيرات حدودية .

ز - المتغيرات : جداول بدون شروط إلزام

يجب أن نذكر إلى أنه بالنسبة للمتغيرات من نوع « جداول بدون إلزام » .
يجب على المستعمل أن يقوم في برنامجه الثانوي بعمليات التدقيق المناسبة للوصلات
بين مختلف الجداول التي تمتاز بعدم وجود حدود الزامية ، إذا كانت هذه الوصلات
موجودة . قد يكون عادة مفيد ، أن نشير بأن الأمر يتعلق بنفس الإلزام ، كما في المثل
التالي .

مثل رقم 5 - حاصل ضرب لا اتجاهي بين متجهين

كي لا نصيّق إستعمال الدالة SCAL ، نستعمل النوع بدون إلزام V لمتغيراته .

```
type V is array (NATURAL range <>) of INTEGER ;  
function SCAL (V1 : V ; V2 : V) return INTEGER is  
begin
```

يجب التحقق من أن V1 و V2 هما بنفس الحجم

```
end SCAL ;
```

يلتزم من الأفضل أن يكون التدقيق قد تم أوتوماتيكياً إذا كانت مواصفة الدالة
SCAL هي :

```
function SCAL (V1, V2 : V) return INTEGER is . . .
```

ولكن لا شيء يلحنا نفترض ذلك .

ح - خاتمة

بشكل عام ، فإن مفهوم البرنامج الثانوي هو قابل للمقارنة مع ذلك الموجود في
اللغات المستعملة . قد يكون مفترضاً أن تكون طريقة عبور المتغيرات محدّدة بغرض
الحصول على أمان أفضل ، وإمكانية نقل كبيرة من مكتبة وأخرى .

6.2 الرزم

تسمح الرزم بتجميع مجموعة الوحدات (أنواع ، متحولات ، برامج ثانوية ،
الخ) ، هذا التجميع يجب أن يتّبع حسب نظرية المعايير ذات الطبيعة المنطقية ، ولكن من

الناحية العملية يمكن للمستعمل أن يقوم بتجميعها حسب ما يوافقه .

جرى دراسة مفهوم الرزم في اللغات المحمولة من مكتبة إلى أخرى (Alghard ، Euclid ، Gipsy ، CLU ، الخ) وهو موجود في اللغات الحديثة (Ada ، Chill ، Modula ، الخ ..) .

هذا المفهوم يتبع :

- من مناهج التفكير عند تصور البرنامج .
- من تجميع الوحدات عند قسمتها بنفس الطريقة .
- من الوصف المجرد لنوع أو لموضوع .

6.2.1 الوصف

6.2.1.1 التركيبية

تنقسم الرزمة إلى قسمين :

● المواصفة التي تجمع جميع الوحدات التي ترسلها الرزمة ، أي جميع المعلومات الضرورية من أجل استعمال جيد للوحدات المرسل . هذه المواصفة تنقسم بدورها إلى قسمين :

- القسم المبرم ، أي القسم المبلوغ في المستعمل .
- وإحتمالاً قسم خاص يجمع المعلومات الإضافية ، الضرورية للمصرف ، وغير المبلوغة في المستعملة («private part») ،

● جسم الرزم الذي يحتوي على قسمين :

- قسم وصفي يجمع المواضيع الضرورية للتشغيل الداخلي للرزم إضافة إلى الإنشاء المتفرّد لوحدات البرنامج الموصوفة في مواصفة الرزم (برامج - ثانوية ، مهام ، رزم ، الخ) .
- قسم تصفير واعداد يجري تنفيذه في لحظة إنشاء الرزم .

هذا المثل عن الرزم المستخرج من المرجع [MR 7.4.1] :

تمرين رقم 6

package KEY_MANAGER is

type KEY is private ;

NULL_KEY : constant KEY ;

procedure GET_KEY (K : out KEY) ;

function "<" (X, Y : KEY) return BOOLEAN ;

private

type KEY is new INTEGER range 0 .. INTEGER'LAST ;

NULL_KEY : constant KEY := 0 ;

end KEY_MANAGER ;

- قسم مرئي مبلوغ من المستعملين

- قسم خاص بالروزمة

package body KEY_MANAGER is	- جسم الرزمة
LAST_KEY : KEY;	- تصريح مركزي
procedure GET_KEY (K : out KEY) is	
begin	- إنشاء الإجراء
LAST_KEY := LAST_KEY + 1;	
K := LAST_KEY;	
end GET_KEY;	
function "<" (X,Y : KEY) return BOOLEAN is	- إنشاء الدالة
begin	
return INTEGER(X) < INTEGER(Y);	
end "<" ;	
begin	- قسم التصغير والاعداد
LAST_KEY := 0;	
end KEY_MANAGER;	

6.2.1.2 المضمون المفصل لمواصفة الرزمة

مواصفة الرزمة لا يمكن أن تحتوي إلا على تصريحات عن :

○ البرامج الثانوية

فقط رأس البرنامج الثانوي أو اسمه يظهر في المواصفة ، الأجسام المناسبة هي موجودة في جسم الرزمة .

○ الرزم والأعمال

كما بالنسبة للبرامج الثانوية ، الأجسام المناسبة هي موجودة بداخل جسم الرزمة الذي يُغلفها .

تداخل القسمين الرئيسيين للرزمة يمكن أن يؤدي إلى صعوبة بالنسبة للمستعمل . فقد يجد نفسه مضطراً لاستعمال أسماء بوصلات أو الكلمة `use` .

○ المتحولات والنوابت

الرزمة لا تؤدي إلى إدخال أي إلزام أو حماية إضافية للوحدات .

○ الاستثناءات

الإستثناءات ليست حرفياً متصلة بالفعل الذي يُندد بإنارتها . وقد يبدو وكأنها مُضافة إلى الرزم بينما يجب أن تكون في البرنامج الثانوي القادر على إنارتها .

○ أنواعها :

هو القسم الأكثر بحثاً . إضافة إلى التصريحات العادية عن النوع ، من الممكن ، في التصريح عن الرزم ، أن يدخل تصريحات عن الأنواع :

- البسيطة

- الخاصة (private) ،

في هذه الحلقة لا يمكن للمستعمل أن يستفيد من تركيبة النوع . ما عدا من مُبَيَّره .
 وبما لا يوجد في تصرفه مؤثرات محدّدة (إختيار ، تأشير ، الخ) ، ما عدا التخصيص (= :
 ، والمقارنة (=) والعكس (= /) . هذه المؤثرات الأخيرة يمكن أن تُقدّم بعض
 الأخطار لأنها مأخوذة في المعنى المنطقي للمصطلح (مقارنة بته بعد بته لحيزين من
 الذاكرة) . من الممكن أيضاً توقع قيمة بالغلط للمتحوّلات من هكذا نوع .
 - خاص مُحدّد (limited private)

في هذه الحالة لا يحقّ للمستعمل أن يستعمل أي مؤثر محدّد . إضافة لذلك ، فإذا
 كان النوع مركباً من عناصر من نوع خاص محلود ، فالمؤثرات المحددة سابقاً غير مسموح بها
 للموضوعات من نوع مُركّب ، خارج الرزمة وحيث النوع الخاص المحدد هو معرف .
 مثل رقم 7

```
package P is
  type T is limited private;
  type S is record
    CH1 : T;
    CH2 : INTEGER;
  end record;
private
  type T is ...;
end P;
```

يظهر النوع S وكأنه من نوع خاص مُحدّد لأن الحقل CH1 هو نفسه خاص ومُحدّد .
 عمليات التعامل ، وعدم التعامل ، والتخصيص بين متحولتين V1 و V2 من نوع S1 ud
 CH12 V1. ygx hgymsù udu hgyldghj ud llmkn ygx V2.CH2 و ' fuh ' lsl, o udu .
 في حالة الأنواع الخاصة (أي عندما يرغب المستعمل بالعمل بشكل مستقل عن
 تركيبة الأنواع المعتملة) ، تحدّد تركيبات الأنواع إذن وبشكل دقيق في القسم الخاص
 للمواصفة .

6.2.1.3 تداخل الرزم

يمكن للرزم أن تظهر في مستويات متداخلة من البرنامج .

يجب أن نلاحظ نقطتين :

- يُعتمد التصريح عن الموضوع في مواصفة الرزمة بسبب فكرة التقسيم وليس نتيجة لفكرة
 الحماية .

- في حالة الرزمة المتداخلة في وحدة أخرى ، فقاعدة القدرات هي التي تحدّد الرؤيا .

هذا هو واضح في المثل التالي .

مثل رقم 8

```

procedure R is
  I: INTEGER := 0;
package P is
  ...
end P;
...
package body P is
  ...
begin
    I := I + 1;
  end P;
begin --corps de R
  I := I + 2;
end R;

procedure R is
  package P is
    I: INTEGER := 0;
    ...
  end P;
  ...
  use P;
  package body P is
    ...
  begin
    I := I + 1;
  end P;
  begin --corps de R
    I := I + 2;
  end R;

```

لهذين الاجرائين نفس الفعل ، بالرغم من تغير موقع التصريح عن I . قاعدة الفدرات هي سائدة في المثل لجهة اليسار ؛ I هي إذا مرئية في الرزمة P . وفي المثل لجهة اليمين ، الرزمة P لا تحمي I التي تقلد على بلوغها مباشرة بالرغم من وجود use ، فهي قابلة للتغيير في جسم R .

● بالنسبة للرزم المتداخلة في وحدات برجة أخرى ، نلاحظ إن :

- الإرسال هو واضح بسبب الفصل في النص بين الموصفة وجسم الرزمة نفسها .
- الإدخال هو ضمني بالكامل ؛ ويجب عبور جسم الرزمة لاييجاد جميع المواضع الواردة ، كل من هله المواضع ينتمي بالضرورة إلى مفهوم التصريح .

● تداخل الرزمة في أخرى يمكن أن يبدو مستغرباً . ويعتمد في حالتين :

- تداخل في نفس الجسم .

تداخل الرزمة في جسم رزمة أخرى يتناسب مع الخطوات المتتالية في تحليل المسألة .

- التداخل في الموصفة .

تداخل الموصفات يمكن أن يحل التنازع على تقسيم الوحدات ، مثلاً الدوران بين

الرزم كما في المثل التالي :

مثل رقم 9

لنفترض ثلاث رزم P3, P2, P1

```

package P1 is
  type A1...;
  ...
end P1;

package P2 is
  type A2...;
  ...
end P2;

package P3 is
  type A3...;
  ...
end P3;

```

فلنفترض إن الرزمة P1 تستعمل النوع A2 ، والرزمة P2 تستعمل النوع A3 والرزمة P3 تستعمل النوع A1 . في هذه الشروط ، ليس من الممكن وضع الجمل with P2 ، with P3 ، with P1 على التوالي أمام P3, P2, P1 لأنه لا يوجد أي نظام ترتيب ممكن للتصريف لهذه المواصفات الثلاث للرمز .

الحل هو في تغليف هذه الرزم الثلاث في رزمة واحدة P :

مثال رقم 10

P - تحتوي على الوحدات المقسومة P3, P2, P1

```
package P
--
type A1...;
type A2...;
type A3...;
package P1... end P1;
package P2... end P2;
package P3... end P3;
end P;
```

ملاحظة :

في حالة الرزم غير المتداخلة في وحدة برجة أخرى ، والمصرّفة بشكل منفصل ، يجب الإشارة بشكل واضح إلى لائحة الوحدات المصّرّفة الواردة ، بواسطة الجملة with .

6.2.2 التقدير

6.2.2.1 التصميم

● عمليات تصميم الأقسام الوصفية وجسم الرزمة هي عمليات مستقلة إلا أن القسم الوصفي يُصمّم قبل الجسم .

تصميم الجسم يتم ، عندما يصبح وجوده ضرورياً (مثلاً ، عند دعوة البرنامج الثانوي المنقول بواسطة هذه الرزمة) . ويقوم على تصميم التصريحات الداخلية ، متبوعة بتنفيذ قسم الأعداد ، إذا كان موجوداً .

● وجود قسم الأعداد يتم تنفيذه عند تصميم جسم الرزمة :

- القسم الوصفي يمكن أن يصبح خليطاً من التصريحات والتعليمات ، وهذا ما نرغب اللغة بتفاديه .

مثال رقم 11

```
package body P is
I: INTEGER;
GET (I);
...
end P;
```

- برنامج غير صحيح لأن القسم الوصفي لم يتم بعد

```

package body P is
  I : INTEGER ;
  package BIDON is
  end BIDON ;
  package body BIDON is
  begin
    GET (I) ;
  end BIDON ;
  ...
end P ;

```

- برنامج صحيح

- عند إنشاء وتصميم الرزمة ، تنفيذ قسم الإعداد والتصغير يمكن أن يؤدي إلى تأثيرات
تحقق من إمكانية قراءة البرنامج . وفي المثل التالي ، كل إنشاء لنموذج عن الرزمة يؤدي
إلى تعديل التحولة الخارجية عن الرزمة X .

مثل رقم 12

```

declare
  X : INTEGER := 0 ;
  generic ( . . . ) package P is
  ...
  end P ;
  package body P is
  ...
  begin
    X := X + 1 ;
  end P ;
  ...
  PP1 is new P ( . . . ) ;
  ...
  PP2 is new P ( . . . ) ;

```

- متحولة مركزية

- تعديل x

- تعديل y

هذا المثل يمكن أن يخدم في حساب عدد النماذج المنشأة للرزمة P . من المؤسف أن
نتمكن من القيام بهذا الحساب خلال طفور التصميم الذي ليس هو الطور الحقيقي
للتنفيذ .

6.2.2.2 قسم خاص - نوع خاص

القسم الخاص ليس له أي مبرر إلا للسماح بحساب حجم الأنواع المصرح عنها في
القسم المرتبي للمواصفة . هذا يُسهّل تعريف الوحدات التي تستعمل هذه المواصفة
وذلك بالسماح بحجز مباشر للمساحة الضرورية من الذاكرة لموضوع أو فرض من نوع
خاص . ومن الممكن أن نأسف لأن يكون إختيار التركيبة المعينة للمعطيات المناسبة لنوع
مجرد يجب أن يتم في مستوى المواصفة لاسباب تتعلق بالتعريف ، بينما من الناحية المنطقية

هكذا إختيار يجب أن يتم ، كما بالنسبة للخوارزميات المناسبة للعمليات على هذا النوع ، في حدود الإنشاء .

وقد جرى تصوّر حلول أخرى : وبالتحديد تلك الخاصة بالزجلة 2 أو التفصيلات في تركيبة الأنواع . هذا الحلّ يؤدي إلى استعمال مؤشر لكل موضوع من نوع معين . ويؤدي إلى خسارة (طفيفة) في الفعالية ناتجة عن عدم التأشير . وفي المقابل ، فهو يناسب بشكل أفضل فلسفة الأنواع المجردة لأن إختيار تركيبة النوع يتم في لحظة تصوّر الزجلة .

6.2.2.3 مختلف إمكانيات الاستعمال

الرزم هي وسيلة عامة . سنعتلي الآن نظرة عامة لمختلف إمكانيات إستعمال هذه الطريقة في البناء :

- كالتريفة COMMON FORTRAN (المحسنة) :

يمكن أن نحصل على رزم لا تحتوي إلا على القسم مواصفات والذي يجمع مجموعة المتحويلات ، الثوابت والأنواع .

- كما يتم تغليف النوع المجرد (مثل رقم 14) :

في هذه الحالة ، يكمل النوع في الخاص من المواصفة . من الممكن أن نستعمل مؤثرات بين المواضيع وهذا النوع . وفي المقابل ، عملية إخفاء تركيبة النوع ليست كاملة ،

- مثل الموضوع المجرد :

المواصفة لا تجمع إلا البرامج الثانوية . للموضوع نفسه يوجد في جسم الرزمة . فلنلاحظ أيضاً أنه إذا إتبعنا هكذا رزمة للكلمة - المفتاح generic ، نحصل على نوع مجرد حقيقي .

- كعنصر من ربيطة إجراءات :

المواصفة لا تحتوي إلا على البرامج الثانوية المجموعة لسبب معين غير منطقي بالضرورة (نفس التطبيق ، نفس المؤلف ، الخ)

- كإجراء للتصغير والإعداد :

يمكن أن تحتوي الرزمة على مواصفة فارغة . في هذه الحالة ، الجسم لا يخدم كثيراً وفقط قسم التصغير والإعداد يمكن أن يكون له فائدة : هذا القسم يُنفذ عند تصميم الرزمة (أنظر المثل رقم 11) .

خاتمة

الرزمة Ada هي وسيلة للبرمجة الزجالية بهدف عام . وهي تسمح بتنفيذ التعريف بشكل سهل يؤمن تماسك مجموعة الوحدات المنقولة . هكذا ، كما برهنا في الأمثلة

السابقة ، المعنى الغام للواسطة يسمح ببعض عمليات الاستعمال الخطيرة . وبشكل خاص ، قسم الأعداد يجب أن يتم تنفيذه بحذر .

6.3 أمثلة كاملة .

6.3.1 مثل رقم 13

الرمزة تستخدم لفرز جدول من الأعداد الصحيحة .

```
package TRI_TABLEAU_ENTIER is
  type TABLEAU_ENTIER is array (INTEGER range <>) of INTEGER ;
  -- نوع غير إلزامي
  procedure TRI (T : in out TABLEAU_ENTIER) ;
end ;

package body TRI_TABLEAU_ENTIER is
  function RANG_DU_MAXIMUM (T : in TABLEAU_ENTIER)
    return INTEGER is
    -- دالة داخلية في جسم الرزمة التي تبحث عن الرتبة العليا لعناصر الجدول
    M : INTEGER := T'FIRST ;
  begin
    for I in T'FIRST + 1 .. T'LAST
      loop
        if T(I) > T(M) then M := I ; end if ;
      end loop ;
    return M ;
  end RANG_DU_MAXIMUM ;

  procedure TRI (T : in out TABLEAU_ENTIER) is
    J,A : INTEGER ;
  begin
    for I in reverse T'LAST .. T'FIRST + 1
      loop
        J := RANG_DU_MAXIMUM (T(T'FIRST .. I)) ;
        A := T(I) ;
        T(I) := T(J) ;
        T(J) := A ;
      end loop ;
    end TRI ;
  end TRI_TABLEAU_ENTIER ;
```

6.3.2 مثل رقم 14

هذا المثل مستوحى من المساعد [MR7.3] ، ولكنه يقدم بعض الفروقات .
الرمزة تنقل النوع RATIONNEL (نوع خاص محدود) ، وجميع الدوال تسمح باستعماله .

package NOMBRES_RATIONNELS is

```

    type RATIONNEL is limited private ;
    function CREER (N : INTEGER ; D : NATURAL)
        return RATIONNEL ;
    function "+" (X,Y : RATIONNEL) return RATIONNEL ;
    function "*" (X,Y : RATIONNEL) return RATIONNEL ;
    function "=" (X,Y : RATIONNEL) return BOOLEAN ;

```

- الملائم = زائد. في الجملة لأن متغيراته هي خاصة ومحددة .

private

```

    type RATIONNEL is

```

```

        record

```

```

            NUM,DEN : INTEGER ; --

```

- المستعمل ليس له بلوغ إلى التركيبة من نوع

```

            RATIONNEL

```

```

        end record ;

```

end NOMBRES_RATIONNELS ;

package body NOMBRES_RATIONNELS is

```

    procedure MEME_DENOMINATEUR (X,Y : in out RATIONNEL) is

```

- إجراء داخلي في الجسم يخفف الكسور في نفس القسوم عليه .

```

    begin

```

```

        X.NUM := X.NUM * Y.DEN ;

```

```

        Y.NUM := Y.NUM * X.DEN ;

```

```

        X.DEN := X.DEN * Y.DEN ;

```

```

        Y.DEN := X.DEN ;

```

```

    end MEME_DENOMINATEUR ;

```

```

    function CREER (N : INTEGER ;

```

```

        D : INTEGER range 1 .. INTEGER'LAST)

```

```

        return RATIONNEL is

```

```

    begin

```

```

        return (N,D) ;

```

```

    end CREER ;

```

```

    function "+" (X,Y : RATIONNEL) return RATIONNEL is

```

```

    begin

```

```

        return (X.NUM * Y.NUM, X.DEN * Y.DEN) ;

```

```

    end "+" ;

```

```

    function "+" (X,Y : RATIONNEL) return RATIONNEL is

```

```

        U,V : RATIONNEL ;

```

```

    begin

```

```

        U := X ;

```

```

        V := Y ;

```

```

        MEME_DENOMINATEUR (U,V) ;

```

```

        return (U.NUM + V.NUM, U.DEN) ;

```

```

    end "+" ;

```

```

    function "=" (X,Y : RATIONNEL) return BOOLEAN is

```

```

        U,V : RATIONNEL ;

```

```

begin
  U := X ;
  V := Y ;
  MEME_DENOMINATEUR (U,V) ;
  return U.NUM = V.NUM ;
end "=" ;
end NOMBRES_RATIONNELS ;

```

الفصل السابع

المدى وإمكانية الرؤية

سنهتم في هذا الفصل ، بطريقة تسمية المواضيع والأغراض ، وبمختلف المشاكل الناتجة عن ذلك [MR8] . لهذا ، من الضروري تحديد مدى التصريح ، الذي يُمثل توسيع البرنامج حيث لهذا التصريح معنى . هذا التصريح يربط المعروف بوحدة معينة . وهذا المعروف يمكن ألا يكفي للإشارة إلى الوحدة (مشكلة الرؤية) . وبشكل خاص ، عندما يتم استعمال نفس المعروف للدلالة على عدة وحدات ، فالإبهام الناتج من مجانسة كهذه يمكن أن تتم إزالته بواسطة التقييع (maskage, hiding) أو تجري المحافظة عليه بواسطة الحمل الزائد surcharge (overloading) ، يتعلّق ذلك إذا بتعدد المعاني (polysémie) . قد نجد عدة طرق للدلالة على نفس الوحدة (إعصاة تسمية (renommage) .

إضافة لذلك ، سنرى إن المواضيع يمكن أن تكون موجودة دون أن تكون مبلوغة في أقسام من النص ليست في مدى التصريح عنها (مواضيع باقية) . وفي النهاية ، سنطرق مشاكل حماية هذه المواضيع .

7.1 مدى التصريحات

7.1.1 الصيغة

بشكل عام ، يبدأ مدى التصريح من الإلتقاء الأول لهذا التصريح ، ويمتد حتى نهاية التركيبية التي يظهر فيها . نجد في لغة أدا المفهوم العادي للمدى . وينحصر فعلياً بهذا القسم فقط من النص عندما تكون التركيبية هي فترة أو جسم (برنامج ثانوي ، رزمة أو مهمة) . ويمتد أيضاً إلى التصريح الضمني لمتغيرة الحلقة ، التركيبية هي الحلقة نفسها . الصيغة [MRA] تحمل من وجهة النظر هذه ، دقة أكثر . يبدأ المدى من المكان الذي يظهر فيه المعروف في المرة الأولى في التركيبية ، هكذا فهذا المعروف لا يمكن إستعماله كإسم إلا في نهاية التصريح .

وفي الحالات الأخرى ، يتوسع المدى إلى كامل أو إلى قسم من مدى التركيبة نفسها . وهذا قد يوجز في الجدول رقم 1 .

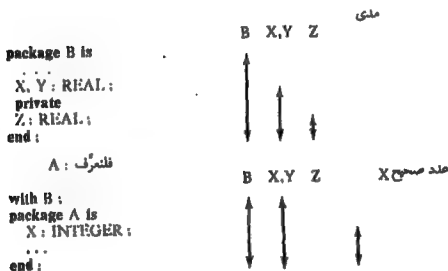
مكان التصريح	توسيع المدى
اللقسم للأولي من الرزمة اللقسم الخاص من الرزمة مواصفة المهمة مركب التركيبة تجزئ متغير شكلي حرف ترقيمي	مدى الرزمة جسم الرزمة مدى المهمة مدى تصريح عن نوع التركيبة مدى الوحدة حيث هي شكلية مدى النوع الترقيمي

جدول 1 - توسيع في مدى التصريح

إضافة لذلك ، فوحدات الرييلة (رزم و برامج ثانوية) يمكن أن ترى مداها موسعاً إلى وحدة تصريح A مختلفة بواسطة الأمر with السابق . وهي تضاف إلى الرزمة النموذجية المختارة في هذا التصريف .

مثال رقم 1

لنفترض وحدة الرييلة B المعرفة كما يلي :



7.1.2 التقدير

هذه التعاريف عن المدى هي كافية بشكل عام . ولكنها ، تبرهن إن التصريحات عن القسم الخاص للرمزة ليست مستعملة إلا في الجسم . هذا القسم لم يدخل إلا في نهاية التصريف . وللإصرار على هذا العمل ، وجب على اللغة أن تحدد هذه التصريحات في هذا القسم بالوحدات المرتبطة فقط بالقسم الخاص المصرح عنه في القسم المرئي .

يجب التذكير إنه إذا كانت الرزمة تعرف الوحدات الداخلة معها (القسم المرئي) ، فلا يوجد أي تأثير على مداها الخاص .

7.2 الرؤيا

7.2.1 صيغة الرؤيا

التعريف يحدد الوحدة ويربطها بمعرف . وفي مجموعة ثانوية من مدى هذا التعريف ، يمكن للمعرف أن يستعمل فقط للدلالة على هذه الوحدة ، ويقال عنه أيضاً أنه مرئي مباشرة . في باقي المدى ، يجب تحديد بعض المفاهيم . هكذا ، فلنفترض معرفاً معيناً I ، مستعملاً لوحدة معينة ، هذا المعرف يتعلق بوحدة وحيدة . مؤلفوا إذا حكموا على هذا الإجماع بشكل قوي في حالتين : الأسماء الحرفية للترقيم والمناهج الثانوية . يقال عن المعرف I أنه محمول بشكل زائد ، إذا كان يتعلق بعدة وحدات ، والنص الكامل يسمح برفع الأهم :

● الإهم الناتج عن التحميل الزائد لإسم حرفي للترقيم ، يجب أن يتم إزالته بواسطة النوع المنتظر من النص الكامل . يجب التذكير بأن الإهم يمكن إزالته بتمييز الإسم الحرفي بواسطة نوع الترقيم الذي يخرج منه ، لهذا السبب لم يكن معرف الإسم الحرفي للترقيم مقنعاً بواسطة التصريح عن الإسم الحرفي للترقيم .

● الإهم الناتج عن المناهج الثانوي (أو عن مؤشر) ، يجب أن تتمكن من إزالته بعد أخذ النص الكامل أو القرينة بالحسبان : أنواع المتغيرات الفعلية ، نوع النتيجة المحتملة . لهذا ، فاللغة تربط قبل أي شيء إسم حرفي للترقيم بدالة بدون متغيرات بسيطة ، والنتيجة هي نوع ترقيمي ينتمي إليه الإسم الحرفي . مع أخذ هذا بالحسبان ، فإن هذين التصريحين عن المناهج الثانوية هما متعادلان إذا كان نظام الترتيب ونوع المتغيرات هوها نفسهما . وإذا كان نوع النتيجة (للدوال أو للمؤثرات) هو نفسه .

لنفترض التصريحين D1 و D2 عن نفس المعرف I في المعطى في المثال رقم 2 . الجدول يحدد ، حسب طبيعة هذه التصريحات عن I ، ما إذا كانت هذه التصريحات في تنازع وتؤدي إلى كون D1 مقنعاً أولاً بواسطة D2 في البناء حيث يوجد D2 ، يجب الإشارة إلى أن المعرف I لا D2 لا يمكن أن يصبح محملاً بشكل زائد إلا عندما يستعمل كإسم ،

أي في نهاية D2 . I التابع لـ D2 هو في جميع الحالات مقنع بداخل التصريح D2 نفسه .
مثال رقم 2

```
declare
  déclaration D1 de I
begin
  ...
  declare
    déclaration D2 de I
  begin
    ...
  end ;
  ...
end ;
```

- D1 هو مقنع أولاً بواسطة D2
- حسب الجدول رقم 2.

D1 \ D2	برنامج ثانوي أو اسم حرفي للترقيم	أخرى
برنامج ثانوي أو اسم حرفي للترقيم	لا إلا إذا كان متعلداً	نعم
أخرى	نعم	نعم

جدول 2 : الحالة حيث التصريحات D1, D2 هما في تنازع ويؤديان إلى تقنع D1 بواسطة D2 .

هكذا فعملية تقنع التصريح لا تمنع أبداً بلوغ الوحدة المناسبة . إذاً ، فالتعبير المؤشر ، والمستعمل كثيراً للدلالة على مركب من تركيبة ، جرى تعميمه في آدا وذلك باعتبار جميع التصريحات كمركبات للإنشاء حيث هو موجود . وفي مدى التصريح ، يمكن للمعرف المضاف I أن يكون محمداً مسبقاً بواسطة معرف البنية C التي تغلفه مباشرة . إذن ، فهذا المعرف I يمكن أن يستعمل لوحدة (وهو مرئي بشكل مباشر) في البناء C إضافة إلى البنى C الداخلة في C والمعلقة كي لا تكون مقنعة بواسطة تصريح I من C . فلنلاحظ أنه في الحالة الأخيرة ، وعلى عكس ما يجري في أكثر اللغات ، عملية تحديد I بواسطة اسم C يسمح بالإشارة إلى التصريح المقنع .

في 7.1.1 ، رأينا أن مدى التصريح الواقع في القسم المرئي للزرمة هو موسع إلى مدى الزرمة . هكذا ، فالمعرف المضاف هو غير مرئي مباشرة ، ولكن يجب أن يكون محمداً

مسبقاً بواسطة اسم الرزمة . لتوضيح التعابير ، نرى إنه من الممكن جعل معرفات القسم الرئيسي للرزمة مرئية مباشرة بواسطة الكلمة use . هكذا ، ففي نقطة معينة من البرنامج ، نرى أن التصريح D عن المَعْرِف I ، الواقع في البناء الذي يُغلف هذه النقطة ، يطفي ، حتى لو كان مقنعا ، ويُمنع أن يكون I المَصْرَح عنه بواسطة D ، في قسم مرئي من الرزمة المذكورة في داخل use ، أن يكون مرئياً بشكل مباشر عندما يكون D ولا في تنازع . إضافة لذلك ، وإذا كان هناك تنازع بين تصريحين عن نفس المَعْرِف في رزم مختلفة مذكورة في جملة use واقعة في إنشاءات تغلف نقطة من البرنامج ، فالمَعْرِف ليس مرئياً بشكل مباشر في هذه النقطة . فلنذكر إن الجملة use لا تصبح فُتَالَة إلا بعد الرمز «:» الذي يُنهي الجملة ، وإن حل أسماء الرزم للجملة لا يأخذ بالحسبان التعديل الجاري على هذا الحل بواسطة الجملة نفسها .

المثال رقم 3 هذه إثبات تأثيرات الجمل use المتتالية ، حسب مختلف الحالات .

مثال رقم 3

```

package A is
  BLANC, NOIR, VERT, VIOLET : INTEGER ;
end A ;

package B is
  type DR1 is (BLEU, BLANC, ROUGE, BEIGE) ;
end B ;

package C is
  procedure MARRON ;
  use B ;
  function BLEU (X : DR1) return BOOLEAN ;
  procedure NOIR ;
  procedure JAUNE ;
end C ;

procedure Q is
  use A, C ;
  use B ;
  type DR2 is (ROUGE, JAUNE, VERT) ;
  MARRON, BEIGE, VIOLET : BOOLEAN ;
  function BLEU (X : DR2) return BOOLEAN is
  end BLEU ;

```

.. NOIR من A و NOIR من C ليست مرئية
 .. BLANC من A لم تعد مرئية
 .. BLANC من B ليست أبداً مرئية
 .. VERT من A ليست مرئية
 .. JAUNE من C و ROUGE من B تبقى مرئية
 .. VIOLET من A و BEIGE من B و MARRON : BOOLEAN
 .. من C ليست مرئية .
 .. BLEU من B و BLEU من C تبقى مرئية .

begin

- فقط BLEU و ROUGE من B

- BLEU و JAUNE من C هي مرئية مباشرة

...
end Q ;

وختارة من بين تلك الداخلة بواسطة الجملة use

استعمال المعرفات المحملة بشكل زائد تعبير معين يجب أن يكون كذلك بحيث
يسمح النص الكامل له بحل هذا المعرف بطريقة واحدة فقط . فلنكمل جسم Q في المثل

: 3

begin

مثال رقم 4

VIOLET := BLEU (JAUNE) ; -- B. BLUE (Q. JAUNE) فقط تفسير

VIOLET := BLEU (BLEU) ;

- يتعلق ذلك بـ C. BLEU (B. BLEU) ، إمكانية الوحيدة مع usc

VIOLET := BLEU (ROUGE) ; - غير مسموح بسبب التفسيرين :

-- Q.BLEU (Q.ROUGE) et

-- C.BLEU (B.ROUGE)

end Q ;

7.2.2 التقدير

يجب قبل أي شيء الإشارة إلى أن الفقرات [MR8.3, 8.4] كانت سبقة التنقيح
ومليشة بالأخطاء . الصيغة [MRA] هي من هذه الجهة أكثر جودة . وفي أغلب
الحالات ، لا تكون الشروحات أعلاه خارجة عن هذه الفقرات ولكن عن المناقشات
المباشرة مع مؤلفي هذه اللغة .

المدى والرؤيا

تقنيع التصريح D1 بواسطة نصريح آخر D2 ، يبدأ من المكان الذي يظهر المعرف
فيه في D2 ، وليس له أي تأثير إلا في مداه .

مثال رقم 5

procedure R is
(a) N : INTEGER ;

- نصريح D1

...
package Q is
(b) I : INTEGER := N ;

- صحيح ، يتعلق ذلك بـ R.N

(c) N : INTEGER := 0 ;

- نصريح D2 : تقنيع N من R

...
end Q ;
end R ;

سنشير هنا إلى الخطأ في استعمال القسم الوصفي للمعروف المقنع داخلياً ، في نفس هذا القسم ، لأن أي تبديل في الخطوط (هنا (b) و (c)) يمكن أن يؤدي إلى تغيير بسيط في الدلالة (أي تغيير في القيمة الأولية I) .

التحميل الزائد

فائدة التحميل الزائد للبرامج الثانوية هي في السماح بإعطاء نفس الاسم إلى برامج ثانوية لا تختلف أبداً في طريقة معالجتها ولكن في نوع المواضيع المعالجة (مثلاً الإدخال الإخراج) . من وجهة النظر هذه ، فإن أولية التحميل الزائد هي بدون أدنى شك كافية أكثر من عمليات التوحيد في الصيغة في لغة Algol 68 ، لأن تحديد البرنامج الثانوي المدعو يتم عند التصريف .

قواعد التقنيع هي متماسكة . البعض يعتبر أكثر منطقية عدم السماح بالتحميل الزائد إلا بين تصريحين عن إجراءات أو بين تصريحين عن أسماء حرفية للترقيم ، والحصول على التقنيع في جميع الحالات الأخرى . هذا هو إختيار محرر وموضح بشكل بدائي في [MRA] وبواسطة تمثيل الاسم الحرفي للترقيم بواسطة دالة بدون متغيرات .

الجملة use

الجملة use هي جذابة ، لأنها تسمح بتسهيل التعابير ، ولكن تشغيلها حرج ، لأن الباقي من البرنامج يمكن أن يجعلها غير فعالة بشكل جزئي أو كلي مما يؤدي إلى خلط كبير بالنسبة للمبرمج .

فلنأخذ هنا الرزمة C كما هي معرفة في المثال 3 ، ولنضعها أمام المثال رقم 6 :

مثال رقم 6 :

```

procedure Q is
  NOIR : INTEGER ;
procedure R is
  function NOIR return BOOLEAN is
  begin
    return TRUE ;
  end NOIR ;
begin
  declare
    use C ;
  NOIR من C غير مرئية بسبب NOIR من Q مع إن هذا الأخير
  هو مفتوح بواسطة NOIR من R الذي لا يوجد أي تنازع بينه وبين NOIR من C
  begin
    NOIR ;
  end ;
end R ;
begin
  ...
end Q ;

```

- غير مسموح

فلنذكر على العكس ، أن هيمنة التصريحات المغنّعة بإنشاءات مُغلّفة تحدّ من أثر مكان الجملة use على فعلها ، مما يعني شيئاً جيداً . هكذا ، في المثال رقم 6 ، فإن تحرّيك الجملة use ، في بداية الإجراء Q لا يُغيّر أبداً في فعله : NOIR من C ليست مرئية في مدى التصريح NOIR من Q .

فلنأخذ المثال رقم 7 . الرزمة B هي مرئية مباشرة بعد السطر (f) ، مما يبرّر كتابة (g) . تأثير هذه الجملة لا يسمح بجعل B مرئية من A.B بسبب التنازع بين B و A . وعلى العكس فهذا التنازع يسحب الرؤيا مباشرة لـ B من A ، والمعرف B ليس مرئياً بشكل مباشر .

مثال رقم 7

```
package A is
package B is
  B : INTEGER ;
end ;
end ;
```

```
use A ;
use B ;
B:=5 ;
BB:=5 ;
```

- يمتلئ ذلك بـ A.B

خطأ : عدد صحيح B مغطى بواسطة A.B

خطأ الرزمة B مغلّطة بواسطة A.B

فلنشير إلى إن إمكانية قراءة البرنامج هي في بعض الأحيان غير مرضية ، لأن إستعمال المعرف ، في غياب التصريح ، يحتاج إلى تحليل جميع الرزم المذكورة في الجملة use . لتفادي هذه السيئات ، كان يجب السماح بالإشارة بشكل واضح ، في الجملة use ، إلى معرفات الرزمة التي نرغب بجعلها مرئية مباشرة ، وإعتبار إن هذه المعرفات تؤدي إلى نفس قواعد الرؤيا كما وكأنا كانت موضوع التصريحات الداخلة إلى موقع الجملة use .

تصريف مفصل للوحدات الثانوية

كجميع وحدات التصريف ، الوحدة الثانوية A المُشكّلة من وحدة A و أم B ، يمكن أن تكون مسبقة بالجملة with . أما الوحدة C (الوحدات) المذكورة في الجملة with فتضاف إلى الرزمة النموذجية ، يؤخذ بالحسبان النص الكامل للجلد A في B . كل شيء يسير كما وكان B يقوم بالتداخل مع C . من الممكن إذا أن تقوم الوحدة B بتقنيع المعرف C الذي سيصبح غير قابل للرؤية بشكل مباشر في A . بينما سيصبح مرئياً بواسطة STANDARD.C مع الاحتياط بأن لا يكون B قد قنّع STANDARD (في هذه الحالة

هو غير مبلوغ بالكامل في A1) . هكذا ، فالتشريف المتفصل للوحدة الثانوية يجب أن يبقى بالنسبة للمبرمج ، وسيلة لتسهيل الإستغلال وليس تقنية في « التقنيع » . فلنشر في النهاية إلى أن جميع التصريحات في الوحدة المخلّفة يمكن أن تُقنّع المعرفة الداخلة بواسطة الجملة use المربوطة في الوحدة - الثانوية وتزيد أيضاً من الغموض بالنسبة للمبرمج .

المتغيرات الشكلية

جرى توسيع مدى المتغير الشكلي - إلى الوحدة حيث يصرّح عنه . هكذا فمعرّف المتغير الشكلي ليس مرثياً (خارج الجسم) إلا في الموقع النحوي لإسم المتغير الشكلي للدعوة هذه الوحدة . يبدو جيداً ، إن الإمكانات المقلّمة هي مفيدة ، وإستعمال التعبير مدى / رؤيا ليس ملائماً بشكل جيد .

7.3 إعادة التسمية

7.3.1 المبدأ

عملية التصريح لإعادة التسمية تسمح بربط إسم جديد إلى الوحدة ، مع المحافظة على الإسم القديم .

المعرّف الداخِل يتبع نفس قواعد الرؤية كما وكأنه يتعلّق بتصريح عن وحدة جديدة . ويمكن أن يتعلّق ذلك بإعادة تسمية بسيطة نصّية تسمح مثلاً بتسهيل عملية الكتابة . ستكون هذه هي الحالة الوحيدة عندما تكون الوحدة هي عبارة عن إستثناء أو رزمة . وعندما تكون الوحدة عبارة عن موضوع ، هو نفسه عبارة عن مُركّب من موضوع آخر ، فهذا يؤدي إلى تصميم جزئي لمعرّف المركّب . هكذا ، فوجود هذا المركّب لا يجب أن يتعلّق بالتمييز . وفي النهاية لا يمكن إعادة تسمية موضوع من نوع مجهول لأنه لا يمكننا كتابة إسم هذا النوع .

مثال رقم 8

```

type T is array (INTEGER range <>) of REAL ;
procedure MAX (A : in out T ; I : in out INTEGER) is
  AI : REAL renames A(I) ;
  Z : REAL ;
begin
  for J in A'RANGE loop
    if A(J) > AI then
      I := J ;
    exit ;
  end if ;
end loop ;
Z := A(I) ;
A(I) := AI ;
AI := Z ;
end ;

```

- إعادة تسمية A (3) إذا كان 3 = 1

- تبديل A(I) مع A(3)

مثال رقم 11 . تعديلات

package B is	(a)
... BLANC بـلون	(b) type DR1 is (BLEU, BLANC) ;
end B ;	(c)
use B ;	(d) -- BLEU et BLANC de B sont vis
package C is	(e)
BLANC : INTEGER ;	(f)
end C ;	(g)
procedure Q is	(h)
use C ; رؤية BLANC من C	(i) BLANC من C و BLANC من B
...	(j) هي مخطئة
begin	(k)
BLANC:=3; صحيح	(l) BLANC غير مسموح لأن أي BLANC
end Q ;	(m) هو مرئي

في المثال رقم 11 ، السطر (i) يجعل BLANC من الرزمة C مرئية مباشرة ، والسطر (1) هو صحيح . فلنعدّل الرزمة B ، بإضافة القسم اليمين للسطر (b) . عند ذلك ، في السطر (1) BLANC من C و BLANC من B ليست مرئية ، مما يجعل هذا السطر غير مقبول . وعمل العكس إذا أبدلنا السطر (i) بواسطة :

BLANC: INTEGER renames C. BLANC ;

السطر (1) سيكون دائماً صحيحاً مهما تكن مواصفات B .
فلنشير في هذه الحالة إلى أن BLANC من B لن تكون مرئية ، في جميع الحالات ، في الإجراء Q .

7.4 مدة حياة المواضيع والأغراض

7.4.1 تقديم

يحدّد إنشاء التصريح عن الموضوع بداية وجوده . نهاية هذا الوجود هي مرتبطة مع نهاية وجود البناء حيث هي موجودة . عندما يكون البناء فادرة ، برنامجاً ثانوياً أو مهمة ، فنهاية وجود الموضوع أو الغرض هي إذاً نهاية تنفيذ القدرة ، البرنامج الثانوي أو المهمة . وعندما يكون البناء هو رزمة ، والتصريح موجود في قسم المواصفة أو في جسم الرزمة ، فنهاية وجود الموضوع هو نهاية وجود الرزمة .

من هذه المواضيع ما يبقى للبرامج الثانوية المحتملة للرزمة ، وتنظيمهما يتم باستعمال مكس في الذاكرة لهذه الغاية .

مهمة المواضيع من نوع بلوغ [MR 3.8] هي إرشاد المواضيع الأخرى المنشأة ديناميكياً ، عند تنفيذ البرنامج ، بواسطة نداء « المُخصّص » [MR 4.8] . هذا الأخير هو

لنفترض النداء مع $I = 3$ ، فالتصريح عن المطر (C) يعيد تسمية A(3) . عند الخروج من الحلقة ، لنفترض إن I لم تتغير وإن I تحتوي على دليل العنصر الأكبر من A(3) الذي يستبدل به في (I, m, n) .

عندما تكون الوحدة عبارة عن برنامج ثانوي ، لإعادة التسمية يمكن أن تسمح ، إضافة لذلك ، بتغيير أسماء المتغيرات الشكلية والقيم بالغلط . لا يمكننا إذاً تغيير ترتيب ، طريقة ، الأنواع ولا الحدود القسرية .

مثال رقم 9 .

```
function PLUS_I (X : INTEGER ; I : INTEGER := 1)
    return INTEGER renames "+" ;
I := PLUS_I (3) ;
I := PLUS_I (3, 2) ;
```

- تعيد القيمة 4

- تعيد القيمة 5

7.3.2 - التقدير

كل تصريح عن إعادة التسمية يقوم بإدخال اسم جديد بدون إلغاء القديم . يتج عن ذلك ظاهرة إنتحال الأسماء (aliasing) مما يعني دائماً و خطراً في البرنامج . هكذا في التدقيق الجاري عند إعادة التسمية يقدم واسطة للمبرمج بالتدقيق ببعض المواصفات المتوقعة . وهذا قد يصبح مفيداً عندما تكون الوحدة المسماة آتية في وحدة مكتبة .

مثال رقم 10

```
with A ;
procedure B is
    N : INTEGER renames A.N ;
    ...
end B ;
```

الجملة with تجعل الوحدة A وكأنها وحدة من المكتبة ، أي رزمة أو برنامج ثانوي . تعليمية إعادة التسمية ، التي تشير إلى A.N تفرض أن تكون A هي إما رزمة تحتوي في القسم المرئي على تصريح عن المعروف N من النوع الصحيح . نفس الشيء لإعادة تسمية البرنامج الثانوي تسمح بالتحقق من نوع وصيغة المتغيرات الشكلية إضافة إلى نوع نتيجة الدوال . قد يكون مؤسفاً أن لا يكون بتصرفنا أوالية تسمح بالمحافظة على القيم القديمة بالغلط .

إعادة تسمية النوع لا يمكن أن نحصل عليها بواسطة هكذا تصريح . في نص المثل أعلاه ، التصريح Subtype T is A.T ، يعني إن الرزمة A هي في القسم المرئي منها عبارة عن تصريح عن النوع (أو النوع الثانوي) T .

مرتبط بالنوع بلوغ ، و مجموعة المواضيع التي ينشئها تُؤلف ما تُسميه اللغة مجموعة والتي تربط بالنوع - بلوغ . يُوجد الموضوع من المجموعة طالما بقي هو نفسه أو أحد مركباته ، مبلوغاً وإذا كان يتعلّق بمهمة أو إذا كان هناك مهمة كمركب ، طالما إن هذه المهمة غير منتهية . عندما ينتهي من الوجود ، يمكن إسترجاع المكان الذي يشغله من الذاكرة . تعريف اللغة يترك لمصمّم المصرّف بعض الحرية في إختيار طريقة إدارة وتنظيم المكان الفارغ . تقدّم اللغة للمبرمج ثلاث أواليات تسمح له ببعض عمليات المراقبة لهذا التنظيم .

- 1 - قد يحدّد المكان الجاهز بالمواضيع التابعة للمجموعة (مواصفة الطول [MR 13.2]) ؛
- 2 - قد يعلم المصّرّف بأنه لا يوجد إستعادة أوتوماتيكية للمكان المشغول بواسطة مواضيع المجموعة التي تصبح غير مبلوغة .
- 3 - قد يطلب بشكل واضح تحرير المكان المشغول بواسطة موضوع ، هذا التحرير يتم بدون تدقيق [MR 13.10.1] .

مثل رقم 12

```

type VOITURE is ... ;
type REP_VOITURE is access VOITURE ;
for REP_VOITURE'S STORAGE_SIZE use
    2000 * ((VOITURE'S SIZE / SYSTEM.STORAGE_UNIT)
pragma CONTROLLED (REP_VOITURE) ;

MA_VOITURE : REP_VOITURE ;
procedure LIBERER is
    new UNCHECKED DEALLOCATION (VOITURE, REP_VOITURE) ;
...

MA_VOITURE := new VOITURE ( . . . ) ;
LIBERER (MA_VOITURE) ;

```

- تحديد بـ 2000 سبارة
 - لا يوجد استعادة أوتوماتيكية
 - إعداد إلى صفر null
 - إنشاء نموذجي لإجراء تحرير الذاكرة
 - بدون تدقيق
 - إنشاء الموضوع المشار إليه بواسطة
 - تحرير المكان المشغول بواسطة الموضوع
 - المشار إليه بواسطة MA_VOITURE
 - والذي يأخذ القيمة null

7.4.2 تقييم

عمليات الاختيار التي تُحدّد مدة حياة المواضيع ، ينتج عنها الحصول ، على قدر ما نستطيع ، عل تنظيم المكان الفارغ على شكل مكتمس . وهذا ما يجري في أغلب اللغات بالنسبة للمواضيع المصّرّ عنها في البرنامج . التركيب على شكل رزمة يسمح بالحصول

عليه بالنسبة للمواضيع الباقية .

بدون تأثير خاص من قبل المبرمج ، فالمواضيع المنشأة ديناميكياً سيتم تخصيصها في كتلة ، ويمكن بواسطة تنفيذ آدا أن تقوم بتقديم مراجع للذاكرة (ومن الواضح بأنه قد لا يقوم به) . فلنشير إلى إن هذا المراجع هو شديد التعقيد ، وباهظ الثمن في الوقت . مفهوم تجميع المواضيع يقترح الفكرة المحتملة بعدم إسترجاع المكان المشغول بهذه المواضيع إلا عندما ينتهي وجود النوع - بلوغ المناسب ، لأنه عند ذلك يمكن الجزم بأن أياً منها مبلوغ .

يمكن للمبرمج أن يفرض هذه الطريقة (Pragma CONTROLLED) من المثل 12) . عندما يقوم المبرمج بتحديد مواصفة طول للنوع بلوغ ، يمكن للمُنفذ أن يحفظ في المكس حيزاً بهذا الحجم من الذاكرة ، يُستعمل بواسطة المُخصّص ، ويتم إسترجاعه وإستعادته ضمناً في نهاية وجود النوع - بلوغ . في النهاية ، التحرير الواضح بدون تدقيق يسمح للمبرمج بأن لا يتم لغياب المراجع ، لقاء احتمال حدوث بعض الأخطاء . بإمكاننا إستعارة هذا المفهوم الذي يرى وجوب تقديم بعض عمليات التدقيق على إدارة وتنظيم المكان المُخصّص للمواضيع المنشأة ديناميكياً للمبرمج . كما ويجب الإشارة أيضاً إلى فقدان التجانس في اللغة التي تقدم ثلاث تركيبات نحوية مختلفة مرتبطة بمراقبة المساحة من الذاكرة : فرائع ، خاصيات ، برامج ثانوية .

التعريف ، بواسطة اللغة ، للحظة تصميم رزم القدرة قبل تنفيذ البرنامج الرئيسي يؤدي إلى كون مواضيع هذه الرزم موجودة خلال مدة تنفيذ البرنامج . وهذا لا يمحصر فقط الإمكانات المحتملة لتغطية (النظام) ، ولكنه يمنع محاولات الاختبار الجزئية للبرامج حيث جميع أجسام الرزمة ليست مكتوبة (هذا قد يتم مع أجسام فارغة يمكن أن يُقدّمها محيط البرمجة) .

قد يكون مهماً أن نكون قادرين على أخذ القرار تاركين لمصمّم المبرف / النظام طريقة تنقيح الوصلة (الرباط) المستعملة (تنقيح ديناميكي للأربطة مثلاً) . فلنشير إلى أن هذا الوجود الدائم سيدفع المبرمج إما إلى نقل تصريحاته عن المواضيع ذات الأحجام الكبيرة من الرزم نحو البرنامج الرئيسي ، أو لتحويل الرزم إلى وحدة ثانوية .

7.5 الحماية

مسألة الحماية ليست متروكة من المساعد في إستعمال اللغة ، ولكن التقسيم الزجلي يحتاج إلى دراسة الإمكانات المقدمة بواسطة اللغة من وجهة النظر هذه .

مثال رقم 13

```
with P2 ;  
package P1 is  
    ...  
end P1 ;
```

فلنشر أيضاً إلى أن إعادة تعريف عملية التعادل بالنسبة للنوع لا تؤدي أبداً إلى إعادة تعريف ضمنية بالنسبة للأنواع المنشأة من خلاله . إضافة إلى إن إعادة تعريف عملية التعادل بالنسبة للمثل 14 لن تؤدي أبداً إلى إعادة تعريف FILE .

وفي الختام ، يمكن أن نأخذ على لغة آدا الغياب الكامل لعملية الإنتقاء في حماية المواضيع من الرزمة . هذه الحماية للكل أو للأشياء ، ستؤدي إلى إجبار المبرمجين الذين يرغبون بالحماية المنتخبة بزيادة عدد البرامج الثانوية في رزمهم .

المفهوم الثاني المهم يتعلّق بحماية المتغيرات الفعلية عند دعوة البرنامج الثانوي . هكذا ، فطريقة العبور (in, out, in out) هي معرفة في مواصفة البرنامج الثانوي ، وأي عملية تدقيق ليست ممكنة في لحظة النداء . إذا كان طبعياً أن يعرف المبرمج المواصفة في لحظة الكتابة ، فمن المؤسف بأنه لا يستطيع مراقبة تأويله في لحظة النداء إلا جزئياً . التدقيق الوحيد الممكن إجراؤه هو فرض الصيغة in بتقديم تعبير كمتغير فعلي . نحن هنا نواجه تراجع مؤسف بالنسبة للصيغة القديمة [MR 79-5.2.1] حيث كان من الممكن إستعمال عدة رموز عند ربط المتغيرات بالإسم (= ، : ، = : ، = :) .

وبدلاً من إستبدالها برمز (=>) واحد بدون تدقيق ، يجب إضافة ما يلي :

=>	بدون تدقيق
:	الصيغة in المنتظرة
=:	الصيغة in out المنتظرة
==:	الصيغة out المنتظرة

وفي النهاية ولتفادي أن يطلب التغيير الداخلي في الصيغة في مواصفة المتغير ، مراجعة جميع عمليات إستعمال هذا البرنامج الثانوي كي نراقب التوافق في الإستعمال ، قد يكون من المؤسف إعادة تسمية البرنامج الثانوي في الوحدات التي تستعمله . في حالة حدوث أي تعديل ، فإعادة تعريف هذه الوحدات ستكشف الأخطاء الواقعة .

7.6 حالة

بالرغم من بعض الانتقادات المهمة ، فالحكم النهائي على النقاط التروكة في هذا الفصل يمكن أن تعتبر إيجابية . سنضع على عاتق آدا :

- توسيع المدى
- الرؤية غير المباشرة (تعبير مؤشر معمم) .
- إعادة التسمية
- التحميل الزائد
- إدارة المواضيع الديناميكية

فلنفترض المثل 13 أولاً . إدخال الرزمة P1 في مدى P2 يضع مشكلة حماية المواضيع P2 في مقابل أفعال P1 . إذا كانت هذه الأفعال هي كاملة في مقابل المواضيع الداخلية لـ P2 (فلن يتم إدخالها) ، فهي غير موجودة بالنسبة للقسم الرئيسي (إلا إذا كانت من نوع خاص) . إذا كان من الواجب أن يكون عند P2 بعض الحذر لجهة P1 ، فيجب التصريح عن المواضيع الحساسة في جسم P2 وتعريف إجراءات استشارة هذه المواضيع . فيكون مهماً أن نتمكن من تعريف المواضيع « النصف - ثابتة » في رزمة ، أي الثابتة في خارج الرزمة والمتحولة في الجسم . هذا يسمح أيضاً بتفادي التعديلات غير المتزامنة لهذه المتحولات في محيط متعدد الأعمال .

هدف الأنواع الخاصة هو تجربة تمثيل أنواع المواضيع خارج الرزمة ، لأن عمليات التخصيص والتعادل تبقى ممكنة التنفيذ . الأنواع الخاصة المحدودة يمكن أن تسمح على العكس للمبرمج بإنشاء أوالياته الخاصة للحماية ، لأن أيّاً من العمليات من غير تلك المحددة في القسم الرئيسي ليست ممكنة . غياب عمليات التخصيص والتعادل على المواضيع من النوع الخاص والمحدودة في خارج الرزمة يؤدي إلى غياب هذه العمليات على جميع المواضيع المبنية مع هذا النوع من المركبات ، لأن التعادل (مثلاً) على المواضيع المنشأة يُحقق التعادل على المركبات . المثل 14 يبرهن الاستعمال الممكن وهكذا مواضيع منشأة :

مثال رقم 14

```
package P is
type POSITION is limited private;
type FILE is
record
  F_POS : POSITION;
  STATUS : INTEGER;
end record;
...
end P;
```

في خارج P من الممكن التصريح عن المواضيع من نوع FILE . لا يمكن استشارة أو تعديل المركب POS من هذه المواضيع إلا في جسم P . وعلى العكس فيمكن استشارة أو تعديل المركب STATUS خارج P . فلنلاحظ هنا أيضاً إن إمكانية تعريف الأنواع « النصف ثابتة » (التخصيص هو غير ممكن بالنسبة للمواضيع من هذا النوع في خارج الرزمة) ستسمح باستشارة المركبات دون السماح بتعديلها .

فلنذكر إن عملية التعادل يمكن أن يُعاد تعريفها بالنسبة للأنواع الخاصة المحدودة إضافة إلى الأنواع القسرية من خلالها (وهذه هي فقط الحالات الوحيدة لاعادة تعريف التعادل) [MR 6.7] .

ولكن منصرفاً في انتقادنا على التحميل الزائد : ويشكل خاص ، نُفكر أنه ، إذا كان ضرورياً بالنسبة لتصريحات من نفس المجموعة (إجراءات أو أحرف ترقيم) ، فهو غير ضروري بين المجموعات وعلى الأخص ، إذا كان يُستعمل بدون عاذير ، فهو يسيء إلى إمكانية الوضوح في قراءة البرنامج . هكلنا حاولنا إثبات جميع النتائج من استعمال الجملة use التي ، إذا كانت مفيدة للمبرمج لتسهيل الكتابة ، فيجب أن تستعمل بحذر شديد ، فقط في المكان الذي لا تسيء فيه إلى إمكانية قراءة البرنامج (رزمة إدخال - إخراج مثلاً) . وفي النهاية ، ومن وجهة نظر الحماية ، وإذا عمل آدا بشكل أفضل من اللغات الأخرى ، فهو لا يزال بعيداً عن عروض وأفكار [Jones and Liskov 78] .

الفصل الثامن

المهام

- إن تقسيم العمل التطبيقي إلى وحدات مستقلة هو من مواضيع الساعة . من الأعمال المعروفة في الحقل نذكر :
- اللغات التي تستعمل برامج المرقاب : [wirth 77, wirth 80] و [Hansen 75 Brinch Pascal] .
 - نموذج الممثل : [Hewitt and Atkinson 77] Plasma .
 - تختلف اللغات الأصلية :
 - ... [Feldman 77] PLITS ، [Daniel and Ingalls 78] Smalltalk .
 - [Hoare 78] CSP و [Brinch-Hansen 78] DP حيث المفاهيم هي الأقرب من تلك المحفوظة للغة آدا .
- كل من هذه الافتراضات السابقة يركز على نموذج وصف . هكذا نموذج يمكن أن يتميز بما يلي :
- طبيعة الوحدات المتصلة فيما بينها .
 - وسائط المزامنة .
 - وسائط التبادل بين الوحدات .
 - طريقة التمييز المتبادلة للوحدات .
 - ومحتملاً ، أخذ مفهوم الوقت بالحسبان .
- في لغة آدا الوحدة المتصلة هي المهمة
- عدد المهام في تطبيق معين هو غير مثبت بشكل مطلق .
 - تؤمن المزامنة بواسطة أوالية موعد قريب من تلك الخاصة بـ CSP ، وفي أقل قياس ممكن ، من DP .
 - يؤمن الاتصال بواسطة إرسال متغيرات نوعية ، في الاتجاهين ، بسبب الموعد (كما في DP) .

التصريح عن المهمة يتم في ثلاثة أوقات :

- 1 - مواصفة نموذج المهمة وتدعى نوع - مهمة (Type tâche) .
- 2 - التصريح عن جسم النموذج ، (corps) .
- 3 - التصريح عن موضوع المهمة (objet tâche) .

تقوم مواصفة النموذج بمهمة إدخال إسم من نوع مهمة مستعمل لاحقاً ، من نفس القسم التصريحي ، للتصريح عن الجسم ، وفي مدى النوع ، للتصريح عن مواضيع المهمة المنشأة على هذا النموذج . تعرّف المواصفة أيضاً أساء المداخل التي تشكل المعرفات الوحيدة المعنية في مدى المهمة . المداخل تسمح بالزامنة وتبادل المعلومات بين المهام .

التصريح عن الجسم المضاف إلى النموذج يُحدّد الخوارزم . وداخلياً ، يمكن أن يتم إنشاء مواضيع للمهمة بواسطة نداء المخصص new .

في الحالة التي لا يوجد فيها إلا موضوع من نوع مهمة معين ، فمن الممكن إستعمال تعبير مختصر : مواصفة النموذج والتصريح عن الموضوع يتمان بالترام . أما النوع - مهمة المناسب فيبقى مجهولاً . يجب الإشارة إلى أن اللغة لا تسمح بهكذا تعبير إلا لنوعين من المواضيع هما : الجداول والمهام .

يمكن أن يتم تصريح جسم المهمة بسهولة . بهذا الخصوص ، يجري تأليف وحدة - ثانوية للتصريف تدخل في وحدة تصريح أخرى [MR 10] .

8.1.1 الأنواع مهمة [MR 9.2]

تقوم مواصفة النوع مهمة بإدخال إسم هذا النوع . هذا الإسم هو قابل للاستعمال ، مع بعض التقييد ، في أي مكان يمكن أن يظهر فيه إسم - نوع . هكذا ، فالنوع مهمة يمكن أن يكون :

- نوع عنصر من فقرة ، أو عناصر من جدول .
- نوع موضوع معي بواسطة البلوغ .
- نوع متغير من برنامج - ثانوي ، مدخل للمهمة أو وحدة شاملة .
- متغير فعلي لوحدة شاملة .

هكذا ، لا يمكن فرض أي إلزام بخصوص النوع أو النوع الثانوي على نوع المهمة . أكثر من ذلك ، يتعلق ذلك بنوع محدد [MRA 7.4.2] . وفي النتيجة ، لا يُسمح بعمليات التخصيص والمقارنة بين مواضيع المهام ، أو مواضيع من نوع مركب التي تحتوي على مهام . يدل كل من المتغير الشكلي والمتغير الفعلي على نفس المهمة ، عندما يتر موضوع مهمة إلى المتغير .

مثال

declare

- مواصفة نموذج المهمة

```

task type RESSOURCE is
  entry ACQUERIR ;
  entry RELACHER ;
end RESSOURCE ;

```

RESSOURCE هي اسم نوع - المهمة
ACQUERIR و RELACHER هي مدخل

R1, R2 : RESSOURCE ;

التصريح عن موضعين مهمة R1 و R2
من نوع مهمة RESSOURCE

```

type A_RESSOURCE is access RESSOURCE ;
A_R : A_RESSOURCE ;
...

```

التصريح عن بلوغ قادر على مراجعة
موضوع من نوع مهمة RESSOURCE

```

task body RESSOURCE is
  ...
begin
  ...
end RESSOURCE ;
...

```

التصريح عن الجسم المضاف
إلى النموذج السابق
القسم الوصفي

إنشاء موضوع مجهول من نوع مهمة
RESSOURCE يُستدل عليه لاحقاً بواسطة
البلوغ A-R

```

begin
  ...
  A_R := new RESSOURCE ;
  ...
  ...
end ;

```

ملاحظات

- مواصفة النوع تسبق التصريح عن الجسم في النص ، كلاهما يجب أن يظهر في نفس القسم الوصفي . يفرض النحو أن تكون التصريحات عن مواضيع المهمة ، وعندما تتم في نفس القسم الوصفي ، سابقة للتصريح عن الجسم . النداءات المتبادلة بين المهام هي أيضاً ممكنة .

- عندما يكون التصريح عن البلوغ لنوع مهمة سابقة للتصريح عن الجسم ، فيجب أن لا يحتوي على إعداد وتصغير (مثلاً ، «A:T:= new TT» يجب أن لا تسبق التصريح عن

الجسم TT) . هكذا ، فمحاولة جعل المهمة فعالة قليل تصميم الجسم المناسب يؤدي إلى حالة إستثنائية ؛ [MRA 9.3] SOME-ERROR ؛ حيث مركبات القسم الوصفي هي مصممة على التوالي حسب ترتيب ورودها في النص [MR 3.0] .

مرور المهام إلى المتغيرات يؤثر كإعادة التسمية مهما يكن نوع العبور .

للإحاطة ببعض هذه التقييدات ، يمكن أن نستعمل مهام معينة بواسطة عمليات نوع بلوغ ، أو مهام عبارة عن عناصر من فقرات أو عناصر من جداول يُستدل عليها بواسطة عمليات بلوغ : تخصيص ومقارنة البلوغ المتعمية إلى نفس النوع هي دائماً مشروعة .

8.1.2 إنشاء المهام [MR 9.1, 9.3, 9.5]

إنشاء موضوع - مهمة خلال تنفيذ البرنامج يتألف من عدة مراحل :

1 - تصميم مواصفة النوع - مهمة

هو ممكن خلال تصميم القسم الوصفي حيث تظهر هذه المواصفة .

الفعل :

- إدخال اسم من نوع مهمة .
- تصميم متتالٍ لتصريحات الدخول (مثلاً ، للفسحة المُجزأة لعائلة إدخال) .
- تصميم مواصفات التمثيل .

2 - تصميم جسم النوع مهمة

يتم ذلك داخلياً ، وفي حدود مصادفته في النص في نفس القسم الوصفي .

الفعل :

- وصل الجسم بالمواصفة .
- لا يجري أي تصميم لمواصفات الجسم خلال هذه المرحلة .

3 - تصميم التصريح عن الموضوع مهمة

وهو يؤدي إلى إدخال العمليات المناسبة (ترتيب سجلات الانتظار لهذا الموضوع) .

الفعل :

- مداخل المهمة تصبح قابلة للتأشير .

4 - جعل المهمة فعالة

حسب التعريف ، فإن جعل المهمة فعالة يقوم على تصميم القسم الوصفي لجسم النوع - مهمة . هذا التصميم يتم تنفيذه عدداً من المرات يعادل مواضيع المهمة المصرح عنها . وتتم بعد تصميم القسم الوصفي الذي يصرح عن هذا الموضوع - مهمة . تنفيذ القسم تعليمة المناسب لا يبدأ إلا عندما تصبح جميع المهام المركزية فعالة .

8.2 تنظيم الوقت [MR 9.6]

تعرض أدا عدداً من الأصول والأنواع تسمح باعتماد الوقت الحقيقي (وقت مقاس بواسطة ساعة) أو الوقت المقلد . الأنواع المحددة DURATION تسمح بمعالجة فترات الوقت في أجزاء من الثانية .

يمكن أن تُعلق المهمة تنفيذها خلال فقرة من الوقت N بواسطة التعليمة delay N ، مما يسمح ببرمجة المهام دورياً . فلنذكر غالباً إن هذه التعليمة لا تأخذ في الحسبان الانحرافات التي يمكن أن تنتج عند طلبيات الساعة في البرمجة في الوقت الفعلي . تعني التعليمة delay N فعلياً : تعليق خلال وقت على الأقل يساوي N ثانية . في حالة الأفعال الدورية ، يجب على المبرمج أن يحافظ على المدة الوسيطة للتعليق .

رزمة الربيعة CALENDAR تعرف النوع الخاص TIME ، الدالة LOCK التي تعطي الساعة الحقيقية من نوع TIME ، إضافة إلى مجموعة الدوال (YEAR, MONTH, DAY, SECOND, MAKE-TIME) والمؤثرات التي تسمح بمعالجة الساعة .

مثال [MR 9.6]

```
declare
INTERVALE : constant DURATION := 60.0 ;
PROCHAINE_FOIS :
    CALENDAR.TIME := CALENDAR.CLOCK
    + INTERVALE ;

begin
loop
    delay PROCHAINE_FOIS - CALENDAR.CLOCK ;
    ...
    عمل دوري
    ...
    PROCHAINE_FOIS := PROCHAINE_FOIS + INTERVALE ;
end loop ;
end ;
```

هذه الطريقة في برمجة العمليات الدورية تؤدي إلى تفادي جمع الانحرافات الناتجة عن الساعة ، بشرط أن تكون مدة كل عملية تكرارية هي أقل من وقت التعليق .

8.3 الاتصال

تعرض لغة أدا شكلين للاتصال بالمعلومات بين المهام تعكس مفهومين في تركيبة وبنية الحسابات ، ويجب معرفتها : قسمة الذاكرة المشتركة وتبادل الرسائل .

المفهوم الأول يرتكز على القواعد العامة للمدى والرؤية والتي تنطبق على المهام . هذه القواعد تسمح بتعيين نفس الموضوع في عدة مهام ، وأيضاً التبادل اللاتزامني

ملاحظة :

جعل مهام نفس القسم الوصفي فعالة يتم حسب الترتيب العشوائي ويمكن أن يكون بالتوازي .

4 مكرر - في حالة وجود موضوع مهمة مُولّد بواسطة المخصص new ، فإنشاء المداخل وتصميم التصريحات عن الجسم يتم بواسطة تنفيذ المُخصّص . هذه الحالة تجمع بشكل واضح إنشاء المهمة بواسطة المخصص new وإنشاء الضمّي لمهام عبارة عن عناصر من فقرة أو من جدول يجري إنشاؤه هو الآخر بواسطة المخصص new .

5 - تنفيذ تعليمات الجسم

يجري إطلاقه في نهاية عملية تفعيل المهمة .

مثلاً :

- الأرقام تعود إلى المراحل الملونة أعلاه :

```

declare
task type TT is
1)   ...
    end TT ;
    type AT is access TT ;
    A, B : AT ;
3)   C, D : TT ;
    package P is
        X : TT ;
    end P ;
task body TT is
2)   ...
    end TT ;
    package Q is
        E : TT ;
3),4),5) F : AT := new TT ;
    end Q ;
    ...
4),5) begin
    3,4,5) if CONDITION then
        A := new TT ;
        B := new TT ;
    end if ;
    ...
end ;

```

-- تصميم المراسفة

-- إنشاء المداخل C و D

-- الشواذ SOMI:ERROR

-- les déclarations de ce corps ne sont pas -- élaborées ici

-- هذا التصريح عن الرزمة هو ضروري إذا أردنا التصريح عن البلوك F وإعداده في نفس الوقت

-- تفعيل المهام C و D بترتيب عشوائي

-- جعل A فعالة

-- أو B

للمعلومات . وهو يسمح أيضاً بالقسمة المحكومة ، بشرط أن تتم برحمة عمليات الزمانة الضرورية . عمليات استثمار اللغة لا تحافظ على تأمين وجود نموذج وحيد لكل متحولة مقسومة ، وتحديداً لأسباب ناتجة عن الفعالية في التركيبات بدون ذاكرة مشتركة . ومصادفة « موعِد » بين مهمتين « تقاسمان » نفس المتحولة يؤدي ، إذا كان ممكناً ، إلى الاستيفاء اليومي إلى واحد من النموذجين ، والذي لم يتغير منذ آخر « موعِد » (إذا تم تعديل كل من النموذجين ، فالنتيجة هي لا متناهية) . نداء أو دعوة نموذج للإجراء النوعي من النموذج SHARED-VARIABLE-UPDATE يؤمن الاستيفاء اليومي للنموذج المركزي للمتحولة المشار إليها كمتغير ، ولكن هكذا عمليات لا تعمل على تأمين هذا الأمر لجميع أنواع المتحولات [MR 9.11] . هذا الشكل في الاتصال هو عرضي في اللغة ، ولا يتطلب أبداً أي تطوير (أنظر 8.8.3.d)

الشكل الثاني للتبادل يركز على مفهوم الموعِد :

يوجد « موعِد » بين المهمة T1 المسماة « داعية » والمهمة T2 المسماة « مدعوة » عندما تدعو المهمة T1 مدخلاً من T2 وتقبل T2 هذا الطلب . ينتهي الموعِد عندما يتم تنفيذ الأفعال المرتبطة بتعليمة مَحْدَدة (accept) للمهمة المدعوة . كل مهمة تعاود تنفيذها بشكل مستقل عن الأخرى .

أولية الموعِد

- تأمين الاتصال المتزامن للمعلومات النوعية بين المهام .
- تسمح بحل مختلف مشاكل الزمانة (تعاون وتوافق) .

8.3.1 نقاط الدخول - التعليمة accept [MR 9.5]

يصرّح عن مداخل المهمة في قسم مواصفة المهمة . مدى هذه المداخل هو نفسه مدى النوع - مهمة . نحريراً ، التصريح عن المدخل هو شبيه بمواصفة الإجراء . ويمكن التصريح عن عائلة مداخل مدونة بمواصفات شبيهة ومحددة بواسطة دليل .

مثال على التصريح عن المداخل

```
task type T is
  entry LIRE ( V : out ELEM );
  entry PRENDRE ;
  entry DEMANDER ( 1..N ) ( D : DONNEE ); -- une famille de N entrées
end T ;
```

دعوة المدخل هي نحريراً شبيهة ببناء الإجراء (ما عدا حالة العائلات) .

أمثلة على نداءات المداخل

- في داخل الجسم T

- نداء المدخل **PRENDRE** للمهمة **T** التي أنتجت هذا المدخل ، منذ هذه المهمة (إغلاق داخلي) أو من خلال إحدى المهام المركزية .
 - في مدى الموضوع **T1** من النوع **T** (وبعيداً في الجسم **T**)
 - نداء المدخل **LIRE** لموضوع المهمة **T1**
 - نداء المدخل ذي المؤشر 4 من عائلة المدخل **DEMANDER** لموضوع المهمة **T1**

المتغيرات الشكلية للمدخل ليست مرئية بشكل مباشر إلا في التعليمات **accept** المذكورة . ويمكن أن تستعمل لأجزاء عمليات إضافة المتغيرات الشكلية / المتغيرات الفعلية عند النداء .

تعليمية قبول الموعد (**accept**) تشير إلى اسم نقطة الدخول ، متغيراتها الشكلية إضافة إلى سلسلة التعليمات المختلفة المطلوب تنفيذها عند أخذ الموعد بالحسبان . عند مواصفة البرنامج الثانوي ، ولنفس المدخل ، يجب أن يكون قسم المتغيرات الشكلية متطابق في مواصفة النوع مهمة وفي التعليمات **accept** التي تظهر في جسم المهمة . وإذا جرى تحديد القيم بالغلط ، تجري حساباتها في كل طلب للموعد .
أمثلة على التعليمات accept

accept PRENDRE ; مزامنة صافية
accept LIRE (V : out ELEM) do - تبادل متزامن
V := LOCAL_ELEM ; - ينتهي الموعد في كل مرة يتم فيها إجراء تعليمية التخصيص
end LIRE ;

التعليمية **accept** التي تخص أحد المداخل لا يمكن أن تظهر إلا في جسم (أو في الفدرات المركزية للجسم) للمهمة التي تحدد هذا المدخل ، ما عدا البرامج الثانوية ، الرزم أو المهام المركزية في هذا الجسم . هكذا ، لا يمكن للمهمة أن تقبل المواعيد إلا على مداخلها الخاصة ، مما يؤدي إلى تفادي أية عملية توازي على المداخل ويتيج في هذه الحالة أفعال استثناء متبادلة .

عند أي نداء ، وإذا كانت المهمة المناداة في الانتظار على المدخل ، يتم ، الموعد مباشرة ، ولا يُجرّن الطلب وتوضع المهمة الداعية في الانتظار . قد يحصل في لحظة معينة ، أن تكون عدة طلبات في الانتظار : فهي في هذه الحالة ، مخزنة حسب ترتيب وصولها . قيمة الخاصية **F*COUNT** تعادل الطلبات الموضوعة في الانتظار على المدخل **E** .

عندما تبلغ مهمة معينة التعليمية **accept** ، وإذا كان هناك طلبات في الانتظار ، فيؤخذ الطلب الأقدم موعداً مباشرة ، وإلا ، توضع المهمة في انتظار نداء معين . التعليمية

accept ، وإذا الموعد ، تنتهي عندما يبلغ تنفيذ التعليمة accept نهاية التعليمة end .
 قد يحصل في جسم المهمة ، أن تعني عدة تعليمات accept نفس المدخل . هذه
 السهولة تسمح بإضافة معالجات محدّدة لهذا المدخل ، وذلك حسب الحالة الجارية
 للمهمة .

8.3.2 عدم التحديد - التعليمة select

تحت نفس الاسم select تتجمّع عدة إنشاءات تسمح بتمثيل المواعيد المشروطة في
 المهمة المتأدية أو في المهمة المناداة .

8.3.2.1 التعليمة select (الجهة المناداة)

في مهمة مناداة ، تسمح التعليمة select بتمثيل مختلف أنواع المزامنة : الانتظار
 المشروط ، الانتظار المتعدد ، الانتظار المحدود في الوقت ، أو أيضاً الانتظار مع نهاية محتملة
 للمهمة .

التعليمة select (الجهة المناداة) هي تعليمة تركيبية حيث يتألف كل من مُركباتها من
 مُدافع (garde) ، يمكن أن يكون فارغاً ، متبوعاً بفرع (branche) .

المُدافع garde هو تعبير بولي ، عندما لا يتحقّق ، يؤدي إلى إلغاء الفرع المضاف .

يوجد ثلاث فئات من الفروع : فروع المواعيد التي تبدأ بواسطة تعليمة accept ،
 والفروع المؤقّدة التي تبدأ بواسطة تعليمة delay والفروع ذات الأطراف النهائية المتعلقة
 بالكلمة المحجوزة terminate . التعليمة select (الجهة المناداة) يمكن أن تنتهي بواسطة
 قسم else . يجب أن يحتوي دائماً على الأقل فرعاً من موعد .

مثال على التعليمة select (الجهة المناداة) .

select	- مُدافع
when not OCCUPE ==>	- فرع من موعد
accept PRENDRE do	- نهاية الموعد
OCCUPE := TRUE ;	- نهاية فرع من موعد
end PRENDRE ;	
X := X + 1 ;	
or	- فرع من موعد
accept RENDRE do	- فقدان المُدافع
OCCUPE := FALSE ;	
end RENDRE ;	
end select ;	

تشغيل التعليمة select (الجهة المناداة)

يقال إن الفرع هو مفتوح إذا جرى التحقق بالمُدافع أو إذا كان هذا المُدافع فارغاً .

يقال إن الفرع هو قابل للإجتياز ، إذا كان ذلك يتعلّق بفرع بمجموع مفتوح على الأقل نداء واحد في الانتظار وذلك على المدخل المعتمد : كل خوارزم يفترض إختياراً خاصاً هو غلط .

تنفيذ التعليمة select (الجهة المتأداة) يبدأ بواسطة :

- حسابة وتقييم المدافعين لتحديد الفروع المفتوحة .

- تقييم التعابير السريعة

- حسابة مؤشرات عائلات الادخال .

التشغيل يختلف لاحقاً . من الممكن تمييز تعليمة الانتظار الانتقادي لمواعيد (لا يوجد قسم else) تعليمة المواعيد المباشرة (القسم else الموجود) .

أ - إنتظار إنتقائي للموعد

من الممكن تمييز ثلاث حالات من الانتظار الإنتقائي حسب طبيعة الفروع المفتوحة : الانتظار البسيط ، الانتظار المحدود في الوقت والانتظار الذي يسمح بإنهاء المهمة .

● الانتظار البسيط يناسب الحالة حيث الفروع الوحيدة المفتوحة هي فروع الموعد :

- إذا كان على الأقل أحد الفروع قابلاً للعبور ، فأحد هذه الفروع يتم إختياره بشكل عشوائي . الموعد يؤخذ بالحسبان ، تُنفذ التعليمة accept ، ويعد ذلك التعليمات التالية من النوع .

- إذا لم يكن أي فرع قابلاً للاجتياز مباشرة ، فالمهمة تنتظر حتى النداء الأول لأي من نقاط الدخول المقبولة في فروع الموعد المفتوحة . يدفع وصول النداء إلى الأخذ بالحسبان المباشر للموعد حسب المخطط السابق .

- الانتظار المحدود في الوقت يناسب الحالة حيث على الأقل هناك فرع مؤقت مفتوح . طريقة عمل الانتظار المحدود في الوقت هي شبيهة بتشغيل الانتظار البسيط ، مع فارق وحيد وهو وجود مدة إنتظار قصوى معينة . هذه الأخيرة هي القيمة الدنيا للفترات الظاهرة في الفروع المفتوحة مؤقتاً . وإذا لم يصل أي نداء إلى نقطة الدخول إلى الفرع المقترح قبل هذه المدة ، يتم تنفيذ الفرع المؤقت الذي يدل على هذه المدة .

● الانتظار مع الإنهاء الممكن للمهمة يناسب التعليمة select التي تحتوي على فرع إنهاء مقترح . طريقة التشغيل هي شبيهة بالانتظار البسيط ، إلا أن الانتظار محدود بمدة حياة محيط المهمة .

التعليقات

- لا يمكن أن يوجد أكثر من فرع إنهاء .
- وجود فرع الإنهاء يمنع وجود الفروع المؤقتة .
- التعليمة select تحتوي على فرع إنهاء لا يمكن أن يظهر في فدرية داخلية للنوع مهمة إذا صرحت هذه الفدرية عن مهام أخرى .
- يتم اكتشاف حالة الخطأ عندما لا يوجد أي فرع مفتوح في لحظة تنفيذ التعليمة select بدون القسم else . يطلق في العمل الحالة الإستثنائية SELECT-ERROR .

ب - موعداً مباشراً

عندما يحتوي على قسم else ، فالتعليمة select (الجهة المطلوبة) توضح إن المهمة المطلوبة لا تستقبل موعداً إلا إذا كان ممكناً مباشرة . يجب أن تكون جميع الفروع هي عبارة عن فروع للموعداً . وإذا لم يكن أي فرع قابلاً للعبور مباشرة ، فالقسم else سيتم تنفيذه : وتنتهي التعليمة .

لا يوجد فرع من موعداً مفتوح	على الأقل فرع من موعداً مفتوح ، ولكن لا يوجد أي فرع قابل للعبور	على الأقل فرع قابل للعبور	
else	else	القسم else موجود	
إنهاء نهاية	انتظار موعداً نهاية	موعداً مباشر	فرع إنهاء مفتوح
انتظار الانقضاء	انتظار موعداً محدد	فرع توقيت لفترة	زمنية مفتوحة
مدة قصيرة	في الوقت	فروع موعداً وحيدة	
حالة استثنائية SELECT-ERROR	انتظار موعداً بسيط		

جدول 1 - طريقة عمل التعليمة select

8.3.2.2 التعليمة select (الجهة المتأدية)

التعليمة select (الجهة المتأدية) تغطي إستعمالين : طلب الموعداً مباشرة والنداء المحدود في الوقت .

أ - طلب الموعداً مباشرة [MR. 9.7.2]

التعليمة select (الجهة المتأدية) مع القسم else

إذا كان من غير الممكن أن يتم الموعد مع المهمة المطلوبة مباشرة ، فالمهمة المنادية تقوم بتنفيذ القسم else : لا يوضع في الانتظار ولا يوجد أي طلب لموعد . فلنذكر ، إنه حتى لو لم يتم الموعد ، فالمتغيرات الفعلية للنداء يتم حسابها .

```
select
R. PRENDRE ;
PUT ("R acquise") ;
else
PUT ("R non acquise") ;
end select ;
```

ب - النداء المحدود في الوقت [MR 9.7.3]

(التعليمة select (الجهة المنادية) مع القسم or delay)

هذه التعليمة تقوم بإدخال طلب لموعد طبيعي . إذا ، وفي نهاية الفسحة الزمنية المحددة في القسم or delay ، لم يتم قبول الموعد في ذلك الوقت ، فيجري إلغاء الطلب . المهمة المنادية تنفذ إذا سلسلة تعليمات القسم or delay .

مثال

```
select
R.PRENDRE ;
PUT ("R acquise") ;
or delay 45.0 ;
PUT ("R non acquise") ;
end select ;
```

8.4 مدة حياة المهام

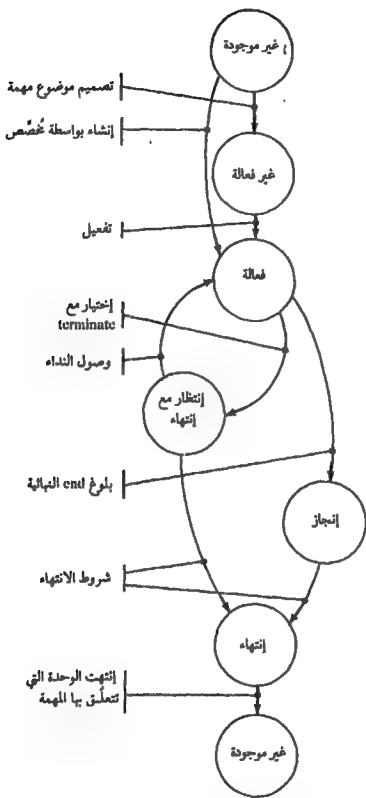
8.4.1 علاقات التبعية

أ - الحالات المصرح عنها

مدة حياة الموضوع هي مدة حياة الوحدة التي تصرّح عنها . هذه القاعدة تنطبق على مواضيع المهمة التي تدعى تابعة للوحدة ومتعلقة بها ، أو تابعة لجسم البرنامج الثانوي أو جسم المهمة التي تصرّح عنها [MR 9.4] .

الحالة الخاصة :

عندما يصرّح عن المهمة بواسطة رزمة داخلية P ، فهي تتعلّق بالوحدة الأولى التي تُخلّف P ، التي تختلف عن الرزمة الداخلية . هكذا ، فمدة حياة كل وحدة مصرّح عنها في القسم الوصفي لجسم الرزمة تعادل مدة حياة الرزمة نفسها [MR 7.3] ، هذه الوحدة



حالات الحياة للمهمة

ليست محدودة بنهاية تنفيذ جسم الرزمة . [MR] تُمَيِّز حالة رُزْم مكتبة البرامج .
إضافة لذلك ، فعندما تتعلّق المهمة بمهمة أخرى ، فهي تتعلّق بالوحدة التي تتبعها
هذه الأخيرة : علاقة التبعية هي إنتقالية .

ب- في الحالات المنشأة بواسطة المُخصّص new
مدة حياة كل موضوع مُنشأ بواسطة المُخصّص New هي نفسها مدة حياة نوع البلوغ
الذي يستقبل التسميات التي يضمها المُخصّص [MR 9.4] . هذه القاعدة جرى تحليدها
لتسمح بتخصيص مواضيع من نفس نوع البلوغ .

المهام المنشأة بواسطة المُخصّص تتعلّق إذاً بالقدرة ، أو بجسم البرنامج الثانوي أو
جسم المهمة التي تصرّح عن النوع بلوغ المستعمل للإنشاء . عندما تصرّح عن النوع بلوغ
في رزمة داخلية P ، فالمهمة تتعلّق بالوحدة الأولى التي تُغلّف P والتي ليست رزمة
داخلية . هذه التبعية هي إنتقالية .

8.4.2 المهام المنجزة والمهام المنتهية

يتم تنفيذ الوحدة - مهمة ، فلدرة أو برنامج ثانوي - عندما تبلغ الأمر End النهائي ،
أو ربما عند مصادفة حالة شاذة . ويتم تنفيذ البرنامج الثانوي أيضاً عندما يجري تنفيذ
التعليمة return .

تنجز المهمة عندما يتم تنفيذ جسم هذه المهمة .
ومنذ لإنجاز المهمة ، يقوم كل نداء على أحد مداخلها بإطلاق الحالة الشاذة
TASKING-ERROR لدى المهمة المنادية .

لا تنتهي المهمة إلا عندما يتم تنفيذ جسمها وعندما تكون جميع المهام التابعة لها إما
منتهية ، وإما بالانتظار في تعليمة select مع فرع إنتهاء مفتوح . عندما تنتهي المهمة ،
تنتهي جميع المهام التابعة والموجودة في الانتظار في تعليمة select مع فروع للانتهاء .
ملاحظة :

ليس للمهمة المنجزة أية فعالية خاصة ، ولكن المهام التابعة هي أيضاً قادرة على
تعديل محيطها . وعندما تنتهي ، تصبح بدون فعالية (مباشرة أو غير مباشرة) على
محيطها ، ولكنها تبقى معنية كأي موضوع .
مثال :

```
task body T is
  A, B : RESSOURCE ;
  type GLOBAL is access RESSOURCE ;
  G : GLOBAL ;
```

- المهام A ، و B هي تابعة لـ T

begin

BL : declare

type LOCAL is access RESSOURCE ;

X : GLOBAL := new RESSOURCE ;

X.all — تتعلق بـ T : وهي معنية بواسطة بلوغ مصرّح
عن نوعه في القسم الوصفي .

L : LOCAL := new RESSOURCE ;

L.all — تابعة لـ T : وهي معنية بواسطة بلوغ بنوع
مصرّح عنه في BL .

C : RESSOURCE ;

C — تتعلّق بـ BL .

begin

G := new RESSOURCE ;

G.all — تتعلّق بـ T

end BL ;

— انتظار الانتهاء من L ومن L.all

end T ;

— انتظار الانتهاء من A, B, G, all ومن X.all

8.4.3 حياة المهمة

كل تصريح عن موضوع مهمة يقوم بإنشاء مهمة جديدة . تركّز المداخل ويمكن أن يتم دعوتها عند إنشاء المهمة . المهمة هي في حالة إنعدام الفعالية . الفعالية الداخلية للمهمة (أنظر 8.1.2) تجعلها تعبر إلى الحالة actif . يجري إطلاق تنفيذ جسم المهمة ، ولكن المهمة المنشأة بواسطة المخصّص new (مباشرة أو في نوع تركيب) هي فعالة منذ وجودها .

عندما يبلغ تنفيذ جسم المهمة التعليمية end النهائية ، تعبر المهمة إلى حالة الإنجاز accompli . وتأخذ الخاصية T*COMPLETED القيمة حقيقة . ولا تترك هذه الحالة إلا عندما تنتهي (أنظر 8.4.2) .

عندما تقوم المهمة بتنفيذ تعليمية الانتظار الانتقائية المحتوية على فرع إنتهاء مفتوح والتي لا تحتوي على أي فرع موعّد قابل للعبور ، تصبح حالتها في الانتظار مع الانتهاء attente avec terminée . وهي تعود إلى الحالة الفعلية منذ أن يدخل النداء على الفرع المفتوح . وهي تمرّ بسرعة بحالة الانتهاء (terminé) ، عندما تكتمل جميع شروط الانتهاء الموجودة . وفي النهاية ، تمرّ المهمة من حالة الانتظار إلى حالة عدم الوجود منذ أن تنتهي الوحدة التي تتعلّق بها المهمة .

8.4.4 التعليمية Abort [MR 9.10]

تؤدي التعليمية Abort إلى إنهاء غير إعتيادي للمهمة المعنية ولجميع المهام المتعلقة

بها . وتؤدي جميع النداءات الجارية أو الداخلية على المداخل إلى إطلاق الحالة الشاذة TASKING-ERROR لدى المهام المتأدية . وتتابع المهام التي هي في طور قبول موعد من إحدى المهام المنتهية بشكل غير إعتيادي تنفيذها بشكل طبيعي بالرغم من موت مرافقتها . طلبات الموعد (غير المقبولة حتى ذلك الوقت) من المهام المنتهية بشكل غير إعتيادي هي ملغاة .

8.5 الانقطاعات [MR 13.5]

أواليات الانقطاع في العتاد ليست قابلة لتكون مرحلة في نموذج واحد . ومع ذلك ، فإن مفهوم الموعد هو عام لكي يسمح بوصول المهام في لغة Ada :

- إما بواسطة أدوات فيزيائية .

- إما بواسطة برامج بمستوى منخفض يؤمن الملقى الضروري يُشبه وصول الانقطاع بنداء على أحد مداخل المهام الخاصة بالمستعمل . هذه اللغة هي مجهزة ببناء يسمح بتخصيص مداخل لفئات الانقطاعات المحددة لعتاد معين . التحديد الوحيد يتعلق بالتغيرات المحتملة للمدخل ، وبشكل ضروري للطريقة in لأن أياً من المعلومات لا يتم إعدادها إلى الإنقطاع .

يتمثل الانقطاع المفقود إذا لم تجري معالجته مباشرة بالتعليمة Select (الجهة المتأدية) مع قسم else ؛ تمثل عملية الانقطاع المخزنة بنداء إدخال بسيط . يختلف أشكال الموعد تسمح ببرمجة ، الجهة المتأداة ، تختلف أنواع المواعيد : قبول مشروط ، قبول مع مدافع ، الخ .

مثال :

task TRAITEMENT DES INTERRUPTIONS is

- تعبير مختصر لتحديد مهمة من نوع مهمة مجهولة

entry NIVEAU_3 (SOUS_NIVEAU : in INTEGER) ;

...

for NIVEAU_3 use at 3 ;

- ربط المدخل بطبقة الانقطاع ورقم 3

end TRAITEMENT_DES_INTERRUPTIONS ;

ربط اسم المدخل بفترة الانقطاع يجب أن يظهر في مواصفة نوع المهمة . ولوافترضنا إن برنامجاً يربط مدخلين بنفس الفترة ، فهو مغلوط . فمن الغلط إذا ، إنشاء أكثر من موضوع من نفس نوع المهمة منذ ربط أحد مداخلها بفترة إنقطاع معينة .

ملاحظة : من غير الممكن ربط نفس المدخل بطبقتين من الانقطاع . لاجراء نفس المعالجة لعدة فئات ، يكفي مناداة نفس الاجراء إلى كل من فروع الموعد لتعليمة select .

يمكن 'حمة مُستعمل أن تنادي مدخلاً مربوطاً بنفس الفئة من عمليات الانقطاع ، مُقلدة بذلك وصول الانقطاع ، ولكن بأولوية أقل من تلك الخاصة بعملية انقطاع حقيقية . (طر 8.6.3) .

8. الأولويات [MR 9.8]

8.6.1 أولوية المهام والبرنامج المركزي

الأمر اللرائعي PRIORITY يسمح بتحديد درجة الإستعمال النسبية للمهام وللبرنامج الرئيسي . عند المهام ، لا يمكن لهذا الأمر أن يدخل إلا في القسم «specification» (مواصفة) : جميع المهام ذات النوع الواحد تمتاز بنفس الأولوية . الأولويات غير المحددة بواسطة أمر ذرائعي (pragma) هي غير محسوبة . مفهوم الأولوية لا يجب ولا يمكن أن يُستعمل لحل مشاكل المزامنة منطقياً . فهو ليس سوى سهولة لتوجيه تنظيم وترتيب المهام المتنازعة .

جميع صيغ اللغة الموضوعية في العمل يجب أن تؤمن وببساطة أنه ، عندما تُنتخب مهمتان بأولويتين مختلفتين ، فتخصص الموارد هو متشابه (مثلاً معالج أو ذاكرة) ، تنفيذ المهمة الأقل أولوية لا يُعاود أبداً قبل تلك المهمة ذات الأولوية الأكبر . لا يوجد أية قاعدة ثابتة بالنسبة للمهام ذات الأولوية نفسها أو بالنسبة للمهام ذات الأولوية غير المحددة . لا يؤمن أي حق في الشفعة (الاسترجاع) : إذا أصبحت إحدى المهام منتخبة بينما هناك مهمة أخرى بأولوية أقل تتمتع بالموارد الضرورية لتنفيذها ، فلا يوجد بالضرورة أي تعليق للمهمة الأقل أولوية .

8.6.2 أولوية المواعيد

تنفيذ الموعد يمكن أن يؤدي إلى زيادة مؤقتة لأولوية المهمة المناداة . هذه الأولوية هي مُعدّلة منذ أن يؤخذ الموعد في الحسبان (وبالتحديد منذ أن يجري إختيار أحد الفروع القابلة للاجتيانز للتعليمية select) . يعاد ترميم الأولوية الداخلية على end الذي ينهي التعليمة accept .

الجدول 2 يحدّد قواعد حساب أولويات الموعد

أولوية المهمة المتأدية خلال الموعد	أولوية المهمة المتأداة قبل الموعد	أولوية المهمة المتأداة
$MAX(P1, P2)$	P2	P1
X عشوائي طالما $X \geq P1$	غير محدد	P1
X عشوائي طالما إن $X \geq P2$	P2	غير محدد
غير محدد	غير محدد	غير محدد

وعلى العكس ، عندما تقوم المهمة بتنفيذ نداء للدخول ، فلا تؤخذ أولويتها في الحسبان .

- فهي دائماً موضوعة في طرف سجل الانتظار المرتبط بالمدخل [MR 9.5] .
- إذا قامت المهمة المناداة بتنفيذ التعليمة select ، يتم إختيار أحد الفروع القابلة للاجتياز عشوائياً ، دون أن يكون قد أخذ بالحسبان للأولويات النسبية للمهام المنادية والموجودة في رأس سجل الانتظار [MR 9.7] .

8.6.3 أولويات الانقطاعات

أخذ العلم بالانقطاع هو حالة خاصة من الموعد حيث يلعب الجهاز الفيزيائي ، بشكل أكثر أو أقل مباشرة ، دور المهمة المنادية . وحسب الإتفاق ، فأولوية هذه الشبه - مهمة هي أعلى من أولوية كل مهمة مبرجة . ينتج عن ذلك إن معالجة الانقطاعات هي بأولوية مناسبة لكل معالجة أخرى : المهمة المناداة تحصل وبسرعة على أولوية الشبه مهمة المنادية .

الأولويات المناسبة لمختلف الانقطاعات لا يمكن أن تثبت بواسطة أمر صرفي (pragma) . وهي تنتهي عادة بقيمة التعبير المستعمل لتعيين فئة إنقطاع في المواصفة التي تربط المدخل بهذه الفئة .

8.7 أمثلة

8.7.1 إتصال مباشر مجهول بدون داريء

يتيح المنتج لأي مُستهلك ، وكل مستهلك يستقبل من أي منتج .

```

task SERVEUR is
  entry PRODUIRE (BUFPRO : in BUFFER);
  entry CONSOMMER (BUFCONS : out BUFFER);
end SERVEUR ;

task type PRODUCTEUR ;
task type CONSOMMATEUR ;

task body SERVEUR is
begin
loop
  select
    accept PRODUIRE (BUFPRO : in BUFFER) do
      accept CONSOMMER (BUFCONS : out BUFFER) do --instructions
        BUFPRO := BUFPRO ;                      accept emboîtées
      end CONSOMMER ;
    end PRODUIRE ;
  or terminate ;
end select ;

```

```
end loop ;
end SERVEUR ;
```

```
task body PRODUCTEUR is
...
BUF : BUFFER ;
...
begin
loop
...
SERVEUR.PRODUIRE (BUF) ;
...
end loop ;
end PRODUCTEUR ;
```

```
task body CONSOMMATEUR is
...
TAMPON : BUFFER ;
...
begin
loop
...
SERVEUR.CONSUMER (TAMPON) ;
...
end loop ;
end CONSOMMATEUR ;
```

ملاحظات

- المهمة « خادم » تضع المنتجين والمستهلكين في اتصال مباشر مجهول .
- المدافع عن إنتهاء الخادم يسمح له بالإنتهاء الأوتوماتيكي مع محيطه .
- على العكس ، إنتهاء المنتجين والمستهلكين يجب أن يكون مبرمجاً بشكل واضح وجلي لأن terminate لا يمكن أن توجد في تنازع مع التعليمات accept في select .

8.7.2 أمثلة على المزامنة الحرة

مخصص الموارد

```
task type ALLOCATEUR is
entry PRENDRE ;
entry RELACHER ;
end ;
```

```
task body ALLOCATEUR is
begin
loop
select
accept PRENDRE ;
accept RELACHER ;
or terminate ;
end select ;
end loop ;
end ALLOCATEUR ;
```

- الموعدتين يشكلان قسماً من نفس الفرع ،
- ويجري تنفيذهما على التوالي .

ملاحظات

- التعليمة select تحتوي على فرعين : فرع لموعد على المدخل PRIOR (متبوع بموعد RELACHER) ، وفرع إنتهاء .

- متتالية النداءات المقبولة هي : * (PRENDRE, RELACHER) .
- النداء الأخير لا يمكن أن يكون سوى RELACHER
- إستثناء متبادل

```
task DONNEES is
  entry MISE_A_JOUR (ELEMENT : in out TYPE_ELEMENT);
end;

task body DONNEES is
  ...
  TABLE_DONNEES : ...
  ...
  begin
    loop
      accept MISE_A_JOUR (ELEMENT : in out TYPE_ELEMENT) do
        --
        -- accès en exclusion aux données
        --
        end MISE_A_JOUR;
      end loop;
    end DONNEES;
```

ملاحظات

- البلوغ للتركيبة المعنية يتم باستثناء متبادل بالنسبة للمهام الخارجية .
- تعالج الطلبات حسب « ترتيب ورودها » .

8.7.3 مثال لبلوغ محكوم بموضوع مقسوم
تخطيط المزامنة « قارئ / متفح مع أفضلية للمتفحين » .

```
package SYNCHRO_LECTEURS_REDACTEURS is
  task type LECTEURS_REDACTEURS is
    entry DEB_LECT;
    entry DEB_REDAC;
    entry FIN_LECT;
    entry FIN_REDAC;
  end LECTEURS_REDACTEURS;
end SYNCHRO_LECTEURS_REDACTEURS;

package body SYNCHRO_LECTEURS_REDACTEURS is
  task body LECTEURS_REDACTEURS is
    NBLECTEURS : INTEGER := 0;

    begin
      accept DEB_REDAC;
      accept FIN_REDAC;
```

- طلب السماح بالقراءة
- طلب السماح بالكتابة
- إشارة نهاية القراءة
- إشارة نهاية الكتابة

- يفرض أن تكون العملية الأولى هي كتابة

```

loop
  select
    when DEB_REDACCOUNT=0 => accept DEB_LECT;
    NBLECTEURS := NBLECTEURS + 1;
  or
    accept FIN_LECT;
    NBLECTEURS := NBLECTEURS - 1;
  or
    accept DEB_REDAC do
      while NBLECTEURS>0 loop
        accept FIN_LECT;
        NBLECTEURS := NBLECTEURS - 1;
      end loop;
    end DEB_REDAC;
    accept FIN_REDAC;
  end select;
end loop;
end LECTEURS_REDACTEURS;
end SYNCHRO_LECTEURS_REDACTEURS;

```

ملاحظات

- من الممكن المحافظة على الرزمة السابقة في الذاكرة ، لاستعمالها في كل برنامج يتطلب هكذا تخطيط للمزامنة .
 - يجب على المبرمج أن يؤمن طريقة الاستعمال الصحيحة . وبشكل خاص ، يجب أن يؤمن :
 - إن كل قارئ (أو مُنقَّح) يدعى DEB-LECT (أو DEB-REAC) قبل القراءة (أو الكتابة) .
 - إن كل مهمة تدعى FIN-LECT (أو FIN-REDAC) قد دُعيت سابقاً باسم DEB-LECT (أو DEB-REAC) .
 - طريقة الاستعمال يمكن أن تتأمن بواسطة رزمة تنشئ مهاماً من نوع LECTEURS-REDACTEURS ، وذلك بعدم جعلها مبلوغة ما عدا الإجراءات التي تطلب المداخل بشكل صحيح . هذه الطريقة هي موضحة في المثل التالي .
- استعمال مخطط المزامنة السابق لمراقبة البليغ إلى موضوع

```

with SYNCHRO_LECTEURS_REDACTEURS;
use SYNCHRO_LECTEURS_REDACTEURS;

generic
  type T_INFO is private;
package GESTION_DE_LIVRES is

```



```

type LIVRE is limited private;
procedure LIRE (L: in out LIVRE; INFO: out T_INFO);
procedure ECRIRE (L: in out LIVRE; INFO: in T_INFO);
private
type LIVRE is
  record
    SYNCHRO: LECTEURS_REDACTEURS;
    --
    --
  end record;
end GESTION_DE_LIVRES;

package body GESTION_DE_LIVRES is

  procedure LIRE (L: in out LIVRE; INFO: out T_INFO) is
  begin
    L.SYNCHRO.DEB_LECT;
    --
    --
    L.SYNCHRO.FIN_LECT;
  end LIRE;

  procedure ECRIRE (L: in out LIVRE; INFO: in T_INFO) is
  begin
    L.SYNCHRO.DEB_REDAC;
    --
    --
    L.SYNCHRO.FIN_REDAC;
  end ECRIRE;
end GESTION_DE_LIVRE;

```

أمثلة على إستعمال الرزمة السابقة

```

--
type L is ...;
package GL is new GESTION_DE_LIVRES (T_INFO => L);
ETAGERE: array (1..10) of GL.LIVRE;
type ACCES_LIVRE is access GL.LIVRE;
AL: ACCES_LIVRE := new GL.LIVRE;
...

```

- يؤدي إلى تصميم 10 مهام

- يؤدي إلى تصميم مهمة

ملاحظة :

النوع LIVRE يحتوي على تصريح عن مهمة وعبور المتغيرات فيها يتناسب مع إعادة تسميتها معها تكن طريقة عبور المتغيرات .

8.7.4 اتصال لا تزامني : معطيات مبلوغة مباشرة بواسطة المهام

```

STRUCTURE_DE_DONNEES: array (1..NB_DONNEES) of DONNEES ;
STRUCTURE_RESULTAT  : array (1..NB_RESULTATS) of RESULTAT ;
CALCULE              : array (1..NB_RESULTATS) of BOOLEAN :=
                        (1..NB_RESULTATS => FALSE);
type INDICE is 0..NB_RESULTATS+1 ;

```

- يجب التحقق من إن CALCULE

```

procedure PROCHAIN (EN_COURS : in out INDICE) is
begin
  EN_COURS := EN_COURS + 1 ;
  while EN_COURS <= NB_RESULTATS
  and then CALCULE (EN_COURS)
  loop
    EN_COURS := EN_COURS + 1 ;
  end loop ;
end PROCHAIN ;

task ALGORITHME_1 ;
task body ALGORITHME_1 is
  EN_COURS : INDICE := 0 ;
begin
  loop
    PROCHAIN (EN_COURS) ;
    exit when EN_COURS > NB_RESULTATS ;
    STRUCTURE_RESULTAT (EN_COURS) := ...
  end loop ;
end ALGORITHME_1 ;

task ALGORITHME_2 ;
task body ALGORITHME_2 is
  EN_COURS : INDICE := 0 ;
begin
  loop
    PROCHAIN (EN_COURS) ;
    exit when EN_COURS > NB_RESULTATS ;
    STRUCTURE_RESULTAT (EN_COURS) := ...
  end loop ;
end ALGORITHME_2 ;

```

ليست مصنوعة إلا في نموذج واحد
- يُميّز الفعالية : PROCCHAIN لا تُنفَّذ
إذا بامتثاء متبادل بواسطة المهمتين ،
هذه الأخيرة تقوم بتنفيذ حسابات مسبقة غالباً .

- حسب المعطيات عند الإدخال
- والنتائج المحسوبة

-- fonction des données en entrée
-- et des résultats déjà calculés

ملاحظات :

● نفترض وجود عدة خوارزميات (2 في هذا المثال) للقيام بنفس الحساب ، كل من هذه

الخوارزميات يُنتج نفس النتائج ، ومدة الحساب تتعلّق بشدة بالمعطيات أو هي عشوائية .

● كل مهمة ، في هذا البرنامج - تتطلب برنامجاً خاصاً ؛ وكل مهمة ، في نهاية الحساب ، تبحث عن الحساب التالي غير المحسوب حتى ذلك الوقت .

● هذا البرنامج هو بدون معنى إلا إذا افترضنا وجود عدد من المعالجات يعادل عدد المهام ، وإن الهدف الذي نبحث عنه هو مدة الجواب .

● هذا البرنامج هو « مغلوط » بمعنى [MRA] لأنه يستعمل متحولات غير متزامنة . نتيجة الحساب هي صحيحة بشكل دائم ومستقلة عن أي تنفيذ .

8.7.5 مشكلة الترتيب (مثال على إدارة الأسطوانة)

المسألة تقوم على بناء خوارزم لإدارة وتنظيم الأسطوانة والذي يُنفّذ حركات ذراع الأسطوانة . المهام المستعملة للأسطوانة تبلغ الرزمة التي تقوم إضافة لذلك بإرسال إجراء لطلب الإرسال . يجب تجهيز رقم المسار والمعطيات المطلوب نقلها كمكتفيات .

```
package DISQUE is
type PISTE is new INTEGER range 0..20;
type DATA is ..... -- autres paramètres
procedure TRANSMET (NUMERO_PISTE : PISTE ; D : DATA);
end DISQUE ;
```

حالة ذراع القراءة يمكن أن تكون مميزة في كل لحظة بواسطة موقعها واتجاه حركتها . نظرياً ، لا يُغيّر الذراع موقعه إلا عندما لا يبقى هناك طلبات تبادل بالنسبة للمسار الجاري .

وبإمكاننا التمييز بين ثلاث حالات :

- إذا كان هناك طلبات إرسال بالنسبة للمسارات الموجودة نحو الأسفل (في إتجاه الحركة) : يتحرك الذراع نحو الأول بينها .

- وإلا ، إذا كان هناك طلبات موجودة لمسارات موجودة نحو الأعلى ، فإتجاه الحركة هو معكوس والذراع يتحرك نحو الأول بينها .

- وإلا ، لا يوجد هناك أي طلب ، والذراع يبقى غير متحرك في الانتظار .

الحل الأول يقوم على التصريح عن مهمة إدارة الأسطوانة التي تحتوي على عائلة أو مجموعة من المداخل ، في داخل جسم الرزمة .

```

task GESTION_DU_DISQUE is
  entry TRANSFERT (PISTE) (D : DATA) ;
end GESTION_DU_DISQUE ;

```

جسم الأجزاء TRANSMET يتم إختزاله إذاً إلى نداء على المدخل ذي الدليل NUMERO-PISTE .

```

procedure TRANSMET (NUMERO_PISTE : PISTE ; D : DATA) is
begin
  GESTION_DU_DISQUE.TRANSFERT (NUMERO_PISTE) (D) ;
end TRANSMET ;

```

جسم المهمة GESTION-DU-DISQUE يحتوي على تعليمة **select** تحتوي بدورها على فرع موعّد مفتوح على المدخل المرتبط بالمسار الجاري ، وعلى قسم **else** يؤدي إلى تغيير في موقع الذراع عندما لا يوجد هناك نداء في الانتظار .

الإجراء CHANGE-POSITION يفحص المداخل السفلى بواسطة الخاصية COUNT ، وبعد ذلك نحو الأعلى لتحديد الموقع الجديد للذراع . من الممكن أن نلاحظ مع هذا الحلّ ، إنه يوجد إنتظار فُعال بلجهة مهمة لإدارة الأسطوانة عندما لا يوجد أي طلب للتبادل : موقع الذراع هو غير متغيّر .

```

task body GESTION_DU_DISQUE is
  type SENS is (BAS, HAUT) ;
  MOUVEMENT : SENS := HAUT ;
  POSITION_DU_BRAS : PISTE := 0 ;
  procedure CHANGE_POSITION is ...
begin
  loop
    select
      accept TRANSFERT (POSITION_DU_BRAS) (D : DATA) do
        ... -- lance l'entrée-sortie physique
      end TRANSFERT ;
    else
      CHANGE_POSITION ;
    end select ;
  end loop ;
end GESTION_DU_DISQUE ;

```

الانتظار الفُعال يمكن أن يتم تفاديه بواسطة تداخل مهمة وسيطة . هذه المهمة تُسجّل طلبات الإرسال الآتية من المهام المُستعملة ، و ، وعلى طلبات للمهمة LANCEUR التي تقوم مهمتها على إطلاق المداخل - المخارج ، وتدير حركات الذراع . وهي تحكّم هوية المسار القادم وتعيد للمهمة CANCEUR عدد الطلبات على المسار .

المهام المستعملة ، وبعد أن يتم تسجيلها عند المهمة الوسيطة ، توضع في الانتظار لدى المهمة LANCEUR (نداء للدخل المهمة LANCEUR المرتبطة بالمسار حيث يجب أن يتم الإرسال) .

المهمة LANCEUR ، وبعد أن تكون قد حصلت من المهمة الوسيطة على رقم المسار وعدد طلبات التبادل على هذا الأخير ، تقبل النداءات على المدخل المرتبط بالمسار . من الممكن أن نلاحظ أنه قد تتداخل طلبات جديدة للتبادل بالنسبة للمسار الجاري وذلك بعد أن تكون المهمة LANCEUR قد حصلت على عدد الطلبات :

TRANSFERT (PISTE_COURANTE)COUNT>NOMBRE_DEMANDES

هذه الأخيرة لن تتم معالجتها إلا في الضربة القادمة ، وموقع النراع لا يتغير إلا عندما لا يوجد طلبات جديدة على المسار الجاري .

في هذين الحلين نصطدم بمشكلة إنهاء المهمة

- في الحل الأول ، المهمة GESTION-DU-DISQUE تدور إلى ما لا نهاية في حلقة مفرغة . إستعمال الأمر **else** يمنع إدخال فرع الإنهاء في التعليمة **Select** .

- في الحل الثاني ، إنهاء مهام الرزمة ليس أوتوماتيكياً . عندما لا يوجد أي طلب للتبادل ، فالمهمة LANCEUR تبقى مغلقة على تعليمة نداء بملخل **DEMANDE-PISTE** ، وليس في داخل تعليمة **Select** « الجهة المناذلة » . من الممكن أن نجعل الإنهاء أوتوماتيكياً . جدوا الحل 1

```
package body DISQUE is
task INTERMEDIAIRE is
entry ENREGISTRE (NUMERO_PISTE : PISTE);
entry DEMANDE_PISTE
(NOMBRE_DEMANDES : out INTEGER ; NUMERO_PISTE : out PISTE);
end INTERMEDIAIRE ;
task LANCEUR is
entry TRANSFERT (PISTE) (D : DATA);
end LANCEUR ;

procedure TRANSMET (NUMERO_PISTE : PISTE ; D : DATA) is
begin
INTERMEDIAIRE.ENREGISTRE (NUMERO_PISTE);
LANCEUR.TRANSFERT (NUMERO_PISTE) (D);
end TRANSMET ;

task body LANCEUR is
NOMBRE_DEMANDES : INTEGER ;
PISTE_COURANTE : PISTE ;
```

```

begin
loop
INTERMEDIAIRE.DEMANDE_PISTE
(NOMBRE_DEMANDES, PISTE_COURANTE);
while NOMBRE_DEMANDES > 0
loop
accept TRANSFERT (PISTE_COURANTE) (D : DATA) do
-- lance l'entrée-sortie physique
end TRANSFERT;
NOMBRE_DEMANDES := NOMBRE_DEMANDES - 1;
end loop;
end loop;
end LANCEUR;

task body INTERMEDIAIRE;
type SENS is (HAUT,BAS);
INVERSE : constant array (SENS) of SENS := (HAUT=>BAS, BAS=>HAUT);
PAS : constant array (SENS) of INTEGER range -1..1 := (HAUT=>1,
BAS=>-1);

COMPTE : array (SENS) of INTEGER := (0,0);
ATTENTE : array (PISTE) of INTEGER := (PISTE=>0);
MOUVEMENT : SENS := HAUT;
POSITION_DU_BRAS : PISTE := 0;

begin
loop
select
when COMPTE (HAUT) + COMPTE (BAS) > 0 =>
accept DEMANDE_PISTE (NOMBRE_DEMANDES : out INTEGER;
NUMERO_PISTE : out PISTE) do
if COMPTE (MOUVEMENT) = 0 then
MOUVEMENT := INVERSE (MOUVEMENT);
end if;
while ATTENTE (POSITION_DU_BRAS) = 0
loop
POSITION_DU_BRAS := POSITION_DU_BRAS + PAS (MOUVEMENT);
end loop;
COMPTE (MOUVEMENT) := COMPTE (MOUVEMENT)
- ATTENTE (POSITION_DU_BRAS);
NOMBRE_DEMANDES := ATTENTE (POSITION_DU_BRAS);
NUMERO_PISTE := POSITION_DU_BRAS;
ATTENTE (POSITION_DU_BRAS) := 0;
end DEMANDE_PISTE;

or
accept ENREGISTRE (NUMERO_PISTE : PISTE) do
if NUMERO_PISTE < POSITION_DU_BRAS then
COMPTE (BAS) := COMPTE (BAS) + 1;
elsif NUMERO_PISTE > POSITION_DU_BRAS then
COMPTE (HAUT) := COMPTE (HAUT) + 1;
else

```

```

COMPTÉ (MOUVEMENT) := COMPTÉ (MOUVEMENT) + 1 ;
end if ;
ATTENTE (NUMERO_PISTE) := ATTENTE (NUMERO_PISTE) + 1 ;
end ENREGISTRE ;
end select ;
end loop ;
end INTERMEDIAIRE ;
end DISQUE ;

```

8.7.6 جدول المهام العاملة في الصيغة Pipe-line

المشكلة تقوم على إنشاء خوارزم أو لائحة مفروزة ومبنية بواسطة نداء للإجراء INSERE . الدالة PRESENT تسمح بمعرفة فيما إذا كان أحد العناصر موجوداً في اللائحة . اللائحة لا تحتوي مرتين على نفس العنصر ، تُقسّم اللائحة على عدة مهام . الحالة الشاذة « فيضان DEBORDEMENT » تنطلق من مهمة تقوم بالإدخال وتؤدي إلى فيضان اللائحة .

```

package LISTE_TRIE is
type DATA is new INTEGER ;
DEBORDEMENT : exception ;
procedure INSERE (D : DATA) ;
function PRESENT (D : DATA) return BOOLEAN ;
end LISTE_TRIE ;

```

وفي النهاية كي يتم السماح ببعض التوازي بين النداءات لـ PRESENT ولـ INSERE ، تمثّل اللائحة بواسطة جدول من الخلايا ، حيث كل خلية هي موضوع مهمة . البرامج الثنائية PRESENT وINSERE يبدأ كل منها بواسطة نداء على مدخل بنفس الاسم للخلية الأولى (0-LA-LISTE) . ينتقل كل طلب من خلية إلى خلية طالما لا يزال يوجد هناك خلية غير مكتملة . المواضيع مهمة من نوع RECEPTION تستخدم كملقى لإرسال نتيجة الخلية الأخيرة التي عاجلت طلباً نحو البرنامج الثانوي المُنادى . حالة كل خلية يمكن أن تكون ممثلة بواسطة زوج مشكّل من العنصر البولي VIDE (فراغ) والصحيح MA-DONNEE . حالة اللائحة يمكن أن تتميز بواسطة المتحولة LIMITE وبشكل يكون فيه لا مُتغيّر النظام هو :

```

I:= (∀ i, 0 ≤ i < LIMITE : not CELLULE (i).VIDE)
and (∀ i, LIMITE < i ≤ MAX_NUMERO : CELLULE (i).VIDE)
and (∀ i,j, 0 ≤ i < j < LIMITE :
CELLULE (i).MA_DONNEE < CELLULE (j).MA_DONNEE
and 0 ≤ LIMITE ≤ MAX_NUMERO + 1

```

في البداية ، اللائحة هي فارغة ($LIMITE = 0$) . تظهر هناك متحولات في اللا متغير ، فقط Ma-DONNEE تظهر بشكل جلي في البرنامج .

```
package body LISTE_TRIEE is
  MAX_NUMERO : constant := 100 ;
  type NUMERO is range 0..MAX_NUMERO ;

  task type RECEPTION is
    entry RES1 (B1 : BOOLEAN);
    entry RES2 (B2 : out BOOLEAN);
  end RECEPTION ;

  type PT_RECEPTION is access RECEPTION ;

  task type CELLULE is
    entry PRESENT (D : DATA ; RELAIS : PT_RECEPTION);
    entry INSERE (D : DATA ; RELAIS : PT_RECEPTION);
    entry INIT (I : NUMERO);
  end CELLULE ;

  type LISTE is array (NUMERO) of CELLULE ;
  LA_LISTE : LISTE ;

  -- Corps de RECEPTION, de CELLULE, d'INSERE et de PRESENT.
begin
  for I in NUMERO loop
    LA_LISTE(I).INIT(I);
  end loop ;
end LISTE_TRIEE ;
```

تستقبل كل خلية رقمها (المدخل INIT) كي تستطيع التعرف على الخلية اللاحقة لها في اللائحة . هذا النوع من مداخل الأعداد والتهيئة هو حلّ كون المهام لا تحمل متغيرات .

كل عملية إطلاق وتفعيل للبرامج الثانوية INSERE وPRESENT تُنشئ موضوعاً من نوع RECEPTION يستخدم كملقى بين البرنامج الثانوي والخلية الأخيرة التي تعالج الطلب . ينتقل الموضوع ، أو بالأحرى بلوغ للموضوع ، من خلية إلى أخرى مع الطلب .

```
task body RECEPTION is
begin
  accept RES1 (B1 : BOOLEAN) do
    accept RES2 (B2 : out BOOLEAN) do
```



```

    B2 := B1 ;
    end RES2 ;
    end RES1 ;
end RECEPTION ;

function PRESENT (D : DATA) return BOOLEAN is
    MON_RELAI : PT_RECEPTION := new RECEPTION ;
    RESULTAT : BOOLEAN ;
begin
    LA_LISTE (0).PRESENT (D, MON_RELAI) ;
    MON_RELAI.RES2 (RESULTAT) ;
    return RESULTAT ;
end PRESENT ;

```

الإجراء INSERE يقوم بإطلاق الحالة الإستثنائية « فيضان DEBORDEMENT » وذلك عندما يستقبل النتيجة FALSE التي تدل على أن الإدخال لا يمكن أن يحصل . تنتشر هذه الحالة الإستثنائية في المهمة المنادية للإجراء INSERE . يستقبل الإجراء الجواب TRUE إذا تم الإدخال بشكل صحيح أو إذا كان العنصر موجوداً في هذا الوقت .

```

procedure INSERE (D : DATA) is
    MON_RELAI : PT_RECEPTION := new RECEPTION ;
    RESULTAT : BOOLEAN ;
begin
    LA_LISTE (0).INSERE (D, MON_RELAI) ;
    MON_RELAI.RES2 (RESULTAT) ;
    if not RESULTAT then raise DEBORDEMENT ; end if ;
end INSERE ;

task body CELLULE is
    MON_NUMERO : NUMERO ;
    MA_DONNEE : DATA ;
    MON_RELAI : PT_RECEPTION ;
    CANDIDAT : DATA ;
begin
    accept INIT (I : NUMERO) do
        MON_NUMERO := I ;
    end INIT ;
    loop
        select
            accept PRESENT (D : DATA ; RELAI : PT_RECEPTION) do
                MON_RELAI := RELAI ;
            end PRESENT ;
            MON_RELAI.RES1 (FALSE) ;
        or

```

```

accept INSERE (D : DATA ; RELAIS : PT_RECEPTION) do
  MON_RELAIS := RELAIS ;
  MA_DONNEE := D ;
end INSERE ;
MON_RELAIS.RES1 (TRUE) ;

```

– phase 2 CELLULE non vide

```

loop
  select
    accept INSERE (D : DATA ; RELAIS : PT_RECEPTION) do
      CANDIDAT := D ;
      MON_RELAIS := RELAIS ;
    end INSERE ;
    if CANDIDAT < MA_DONNEE then
      if MONUMERO = MAX_NUMERO then
        MON_RELAIS.RES1 (FALSE) ;
      else
        LA_LISTE (MON_NUMERO + 1).INSERE (MA_DONNEE, MON_RELAIS) ;
        MA_DONNEE := CANDIDAT ;
      end if ;
    elsif CANDIDAT > MA_DONNEE then
      if MON_NUMERO = MAX_NUMERO then
        MON_RELAIS.RES1 (FALSE) ;
      else
        LA_LISTE (MON_NUMERO + 1).INSERE (CANDIDAT, MON_RELAIS) ;
      end if ;
    else
      MON_RELAIS.RES1 (TRUE) ;
    end if ;
  or
    accept PRESENT (D : DATA ; RELAIS : PT_RECEPTION) do
      CANDIDAT := D ;
      MON_RELAIS := RELAIS ;
    end PRESENT ;
    if CANDIDAT < MA_DONNEE then
      MON_RELAIS.RES1 (FALSE) ;
    elsif CANDIDAT = MA_DONNEE then
      MON_RELAIS.RES1 (TRUE) ;
    elsif MON_NUMERO = MAX_NUMERO then
      MON_RELAIS.RES1 (FALSE) ;
    else
      LA_LISTE (MON_NUMERO + 1).PRESENT (CANDIDAT, MON_RELAIS) ;
    end if ;
  end select ;
end loop ;
end select ;
end loop ;
end CELLULE ;

```

تُمر المهمة Cellule في طورين . خلال الطور الأول ، الخلية هي فارغة . وتجابوب بـ FALSE على جميع الطلبات PRESENT . ومنذ أن تستقبل طلباً للإدخال ، تمر إلى الطور الثاني . لتأمين الحد الأقصى من التوازي ، يجب أن تكون المواعيد قصيرة قدر الإمكان . خلال المواعيد ، نحاول نسخ المتغيرات في التحولات المركزية . لهذا السبب نقوم بعملية بلوغ على الموضوع مهمة وليس الموضوع نفسه .

8.8 التقييم

8.8.1 الجوانب الإيجابية

أ - مفهوم النوع مهمة

يمتاز النوع مهمة بأغلب صفات النوع . وهو ملائم لإمكانية إنشاء المهمة ، إما بالتصريح عنها أو بواسطة تعليمة . فقط نحو التصريحات عن الأنواع مهمة هو قابل للمناقشة : لماذا نُفضل «task type T is» عن التعبير «Type T is task...» ؟

الوصلة بين النوع وجسم المهمة

من الممكن التصريح عن النوع مهمة مشتقة .

مثال

```
task type T1 is ... ;
type T2 is new T1 ;
```

مع إن جسم المهمة لا يُشكّل قسماً من نوعها ، فليس من المسموح ربط الجسم بمختلف أنواع المهمة المشتقة .

مثال

```
task body T1 is ... ;
task body T2 is ... ;
```

غير مسموح به

الأنواع مهمة والأنواع المحدودة الخاصة

الأنواع مهمة هي من الأنواع المحدودة [MRA 9.2] : لا عمليات التخصيص ولا عمليات مقارنة المهام أو المواضيع المركبة التي تحتوي على مهام هي معرفة . التصريح عن الأنواع مهمة في القسم الخاص من الرزم يسمح أيضاً بتقييد العمليات الممكنة .

مثال :

declare

```
package P is
  type T is limited private ;
  procedure OP (TACHE : T) ;
private
```

```

task type T is
  entry E1 ;
  entry E2 ;
end T ;
end P ;

TACHE : P.T ;
type ACC_T is access P.T ;
A_TACHE : ACC_T ;

package body P is
  ...
end P ;

```

```

begin
  TACHE.E1 ;
  P.OP (TACHE) ;
  A_TACHE := new P.T ;
  A_TACHE.E1 ;
end ;

```

- غير مسموح : نقاط دخول TACHE ليست مرئية
 - الدخول ليس قابلاً للتداعٍ إلا من خلال الإجراء P.OP الذي يفرض طريقة للاستعمال
 - لا يمكننا أن نمنع التخصيص
 - غير مسموح

ب - مفهوم الموعد

ميكانكية الموعد تتناسب مع ميكانكية سهلة وقابلة للفهم بواسطة أي مبرمج .
 التشكيل المعتمد هو شديد الفعالية . نُحلّد التعليمات accept المزامنة والتبادل في مرة واحدة ، وهو اختيار غير موجود في جميع اللغات .
 المزامنة

تسمح التعليمات accept بدون متغيّر بالمزامنة فقط . الفعل المعتمد في نقطة المزامنة يمكن أن يكون :
 - إما منفذاً بواسطة المهمة المتأداة بالتوازي مع المهمة المتأدية .
 - إما منفذاً بشكل إجرائي متناسب مع المهمة المتأدية .
 مثال

```

accept E ;
I := I + 1 ;

```

- تنفيذ $I := I + 1$ بالتوازي مع المهمة المتأدية

```

...
accept E do
  I := I + 1
end E ;

```

- تنفيذ $I := I + 1$ بشكل إجرائي

بإمكاننا خلط هاتين الإمكانيتين ، مثلاً :

```
select
  accept E1 do A1 end ; B1 ;
or accept E2 do A2 end ; B2 ;
end select ;
```

المراقبة

التعليمات select و accept تُشكّل ، كما في تعابير المسار ، وسائل تعبير عن التوالي . مثلاً ، طريقة استعمال أحد السجلات يمكن أن يُجَدَّ بواسطة التعبير التالي عن المسار .

```
path ouvrir ; | lire + écrire ; | fermer end path
```

التعبير Ada هو أقل تصوراً :

```
task body FICHIER is
begin
  accept OUVRIER ;
  loop
    select
      accept LIRE (...) do ... end ;
    or accept ECRIRE (...) do ... end ;
    or accept FERMER ; exit ;
    end select ;
  end loop ;
end FICHIER ;
```

على عكس التعابير عن المسار وعدادات الزامنة [Robert et Verjus] فالتعبير عن مراقبة التنفيذ ليس نصاً مفصلاً عن الخوارزم في لغة Ada . وهذا الاختيار هو مبرر :

- العدادات والتعابير عن المسار هي مُتكيفة مع مشاكل التوالي ، وليست متكيفة مع مشاكل الزامنة التي تقوم بإدخال متحولات جلية عن الحالة . هذه الأخيرة تحتاج عادة إلى تعبير إصطناعي ، مثلاً ، إدخال إجراءات فارغة .

- في آدا ، يبقى يمكننا فصل المراقبة بتعريف الرزم التي تحتوي على مهام محدّدة الدور بدقة بتوضيح المراقبة (مثلاً : أنظر 8.7.3) .

الاتصال

طريقة التمييز اللامتوازية للمهام المتصلة - فقط المهمة المنادية تعين المهمة المناداة - هي متكيفة مع كتابة المخطط « زبون - خادم » .

مدى المتغيرات الشكلية لنقطة الدخول هي محدودة بالتعليمة accept . نحصل على

أقصى توازي بين المهام المتصلة بواسطة مواعيد بإجراء نسخ للمتغيرات في التعليمة accept وتنفيذ متتال للفعل الخاص بالموعد .

مثال :

```
select accept E1 (X : T1) do X1 := X ; end ; A1 (X1) ;
or accept E2 (X : T2) do X2 := X ; end ; A2 (X2) ;
end select ;
```

عدم التحديد

تسمح التعليمة select بالتعبير عن عدم التحديد .

مثال

```
task body TAMPON is
begin
loop
select
when PLEINS < N =>
accept PRODUIRE ... ;
or when PLEINS > 0 =>
accept CONSOMMER ... ;
end select ;
end loop ;
end TAMPON ;
```

- Select توضح بشكل جلي
- أنه عندما $0 > \text{PLEINS} > N$
- CONSOMMER أو PRODUIRE
- مقبولة بشكل مختلف

ج - عائلة عائلات المداخل

إمكانية التصريح عن عائلات من المداخل أو مجموعات مداخل لا تُشكّل فقط سهولة في الكتابة . التصريح عن عائلة بجدول N يستبدل N تصريحاً للإدخال . إضافة لذلك ، فقبول الموعد على أحد المداخل حيث يُحسب الدليل ديناميكياً يستبدل تعليمة select بعدد N من الفروع حيث أحد الفروع فقط سيكون مفتوحاً عند كل عملية تنفيذ .

هذا المفهوم هو أكثر أهمية عندما يكون عدد المداخل من عائلة معينة غير محدود إلا بنوع الدليل . الدليل هو إذاً مفيد ليس فقط لتمييز ، مثلاً ، الموارد المتشابهة ، ولكن أيضاً فهو يلعب دور المفتاح ، ليؤمن عدم قبول الموعد إلا إذا أظهر وأرسل المناقدي المفتاح الصحيح .

د - مفهوم المهمة السلبية

نستعمل مفهوم المهمة السلبية لتمييز مهام لا تقوم بتنفيذ سوى تعليمة accept ، داخلة محتملاً في تعليمة select وفي حلقات . من هذه المهام لا تقوم بأي فعل خاص (غير مربوط بموعد) .

التعاون بين المهمة السلبية ومهمة أخرى يسمح بإنشاء بعض محطات لروتينات (مناهج) مُساعدة . عندما تكون عدة مهام في تناقض لنداء إدخال إلى مهمة سلبية ، فالتعليمات accept تنشئ الأقسام الحرجة التي قد تكون شرطية بسبب وجود مدافعين . إضافة لذلك ، فالفائدة من المهام السلبية تأتي من كونها تسمح بإجراء مهمة وتنفيذها بفعالية قريبة من تلك التي تسمح بها اللغات المبنية على مفهوم المراقب [Eventoff et al. 80] .

هكذا ، فلا شيء يفرض كون التعليمات accept هي منفذة بواسطة المهمة المنادية شريطة أن يكون ملحقاً بها طريقة لبلوغ التحولات المعنية بواسطة المهمة المناداة . بعض الاستثمارات تسحب قسماً من هذا العمل لعدم ربط أية «عملية» بالمهام السلبية ؛ البعض الآخر يتقادم التغييرات في الإطار وذلك بتنفيذ التعليمات accept في مفهوم المهمة الجارية (أي تلك الخاصة بمفهومين ، المنادية والمناداة ، الأخيرة الواصلة إلى الموعد) .

8.8.2. الفجوات وعدم الملازمة

أ - النحو

تعلّمة الانتظار الانتقائي هي شديدة الفعالية ، ولكن وصفها النحوي قابل للمناقشة . يمكن للفروع أن تظهر حسب ترتيب معين ، بما فيه تلك المستثناة بشكل تبادلي .

عرض نحو أكثر دقة

```
selective_wait ::=
  select
    [when condition =>] accept_statement [sequence of statements]
  | or [when condition =>] accept_statement [sequence of statements] ;
  last_alternative
  and select ;

last_alternative ::=
  | or [when condition =>] delay_statement [sequence of statements] ;
  | or [when condition =>] terminate ;
  | else sequence_of_statements
```

قواعد النحو هذه (سهلة التحويل في القاعدة LL(1)) ، ومع إنها دقيقة كما في [MR 9.7.1] ، فلا نحدد أبداً الدلالة طالما إن ترتيب تقييم المدافعين هو عشوائي .

ب - المتغيرات والإلزام

المهام هي الوحدات الوحيدة من البرنامج التي لا تخصّها اللغة بمتغيرات نوعية . إضافة لذلك ، فمن غير الممكن تعريف نوع - مهمة بإضافة إلزام إلى نوع أساسي كما في الجداول والفقرات .

مثلاً :

لا تسمح اللغة أبدأ بالتصريح عن النوع «Sémaphore» المتغير بواسطة قيمته الأولية . كنا نتمنى لو نستطيع أن نصريح كما يلي :

```
generic
  VAL_INIT : INTEGER ;
  task type SEMA is
    entry P ;
    entry V ;
  end SEMA ;

  task type SMUTEX is new SEMA(1) ;
  task type SRESSOURCE is new SEMA(50) ;
```

- غير مسموح
- غير مسموح
- غير مسموح

قد يكون من المؤسف ألا نستطيع التصريح ، مثلاً ، عن نوع «Semaphore» بدون إلزام وعن نوع - ثانوي «Scmaphore» باستثناء متبادل .

```
task type SEMA (VAL_INIT : INTEGER) is
  entry P ;
  entry V ;
end SEMA ;
subtype MUTEX is SEMA (1) ;
S1, S2 : MUTEX ;
```

- غير مسموح
- غير مسموح

يقوم الحلّ على إدخال النوع - مهمة في رزمة أساسية مقابل بعض الصعوبة في الكتابة .

```
generic
  VAL_INIT : INTEGER ;
  package P_SEMA is
    task type SEMA is
      entry P ;
      entry V ;
    end SEMA ;
  end P_SEMA ;

  package body P_SEMA is
    task body SEMA is
      VAL : INTEGER := VAL_INIT ;
      ...
    end SEMA ;
  end P_SEMA ;

  ...
  P_MUTEX is new P_SEMA (VAL_INIT => 1) ;
  S1, S2 : P_MUTEX.SEMA ;
```


إضافة النوع مهمة SEMA للدخل ثالث يؤمن الأعداد. هذا الحل ليس مؤكداً ، ولا مُتممياً في الحالة التي يكون فيها المتغير هو (نوع) type .

ج - إتصال دائري

اللغة لا تقدم أي إنشاء مُكرس للتبادل الدائري . الضرورة الجلية لبرمجة هذا النوع من الإتصال يعني عملاً أقل فعالية .

نداء المدخل يعني إما إنتظار المهمة المتأدية حتى تحقيق الموعد ، وإما إعتاق الطلب بعد الإنتهاء من المدة . وإذا رغبتنا بإتصال بدون إنتظار ، مثلاً ، مجموعة رسائل في علبة بريد ، يجب إنشاء مهمة « داري » = Buffer = Tampon « جاهزة دائماً لاستقبال النداءات الآتية من المهام المُرسلة والمُوجّهة . المشكلة تنتج عن إختيار أولية موحدة للمزامنة والتبادل ، هذا الإختيار ليس خراجاً . ويجب الإشارة إلى أنه مهما تكن اللغة ، فالمهمة داري (أو إجراء) هي ضرورية عند مصادفة مشاكل ترتيب الرسائل المركبة كي يتم معالجتها بواسطة التمييز الدقيق لمصفوفات الأولويات المختلفة .

د - إختيار طلبات المواعيد

على عكس PLITS [Feldman 77] ، فلغة Ada لا تسمح بالتعبير عن إختيار طلب الموعد حسب القيم المنقولة . أحد الحلول يقوم ، كما في السابق ، على إنشاء مهمة دائرة تؤمن ترتيب الطلبات . الحل الآخر يقوم على إستعمال دليل في عائلة إدخال لنقل المتغير الذي يقع عليه الإختيار . هذا الحال لا يُعتمد عملياً إلا عندما يكون جدر قيم الدليل ضميماً ، لأنه يجب كتابة فرع لكل قيمة ممكنة .

مثال

```
select
  accept E (PRIOMAX) (...) do PROC ; end E ;
or when E (PRIOMAX) 'COUNT = 0 =>
  accept E (PRIOMED) (...) do PROC ; end E ;
or when E (PRIOMAX) 'COUNT + E (PRIOMED) 'COUNT = 0 =>
  accept E (PRIOMIN) (...) do PROC ; end E ;
end select ;
```

لعدد أكبر من المداخل ، فالحل من المثال التالي هو كاف بصورة أولية ، ولكنه سيثير إنتظار فعال غير مقبول بشكل عام . طالما يوجد على الأقل نداء في الانتظار ، فالنداء الأكثر أولوية يؤخذ بالحسبان . المشكلة هنا هي ، أنه عندما نصل إلى المدخل الأخير من العائلة ، يجب أن يوضع في الانتظار على أول نداء على أي من مداخل العائلة . ولكن ، في آدا ، لا يوجد أية عملية مركزية على أي من عائلة المداخل ، نفتقد إلى الخاصية COUNT التي

تجمع عدد النداءات على كامل العائلة والتعليمية accept على أحد المداخل المختلفة . هذه المشكلة تقع دوماً في خوارزميات الترتيب .

مثال

```
ENTREE := PRIOMAX ;
loop
  select
    accept E (ENTREE) ( . . . ) do
      . . . -- traitement
    end E ;
  ENTREE := PRIOMAX ;
  else if ENTREE = PRIOMIN then
    ENTREE := PRIOMAX ;
  else ENTREE := ENTREE'PRED ;
  end if ;
end select ;
end loop ;
```

كما في المثال الخاص بإدارة ذراع الأسطوانة ، أنظروا هذا الحل الممكن .
- تعني المهمة مختلف النداءات .

- تطلب مهمة الخدمة من مهمة الترتيب ، على أي من المداخل يُوجد الطلب التالي للخدمة .

- يجب على كل طلب أن يشير إلى وصوله إلى مهمة الترتيب التي تعين له على أي من مداخل مهمة الخدمة يجب أن يطلب الموعد .

هـ - تعيين المهام

غياب تعيين فئات المهام

التعيين الجلي لوجهة نداء الدخول يمنع إرسال طلب بالخدمة إلى فئة خدم متعادلة (مثثلة مثلاً بواسطة عائلة مداخل) بشكل يقبل الخادم الحر الأول الطلب .

أحد الحلول يقوم على إرسال نداءات شرطية للإدخال للفحص المتتالي لكل خادم ، لقاء إنتظار فعال للمهام الطالبة ولتحميل زائد غير مفيد لنظام الإتصال .

الحل الآخر يقوم ، مرة أخرى ، على تعريف المهمة الدائرة القابلة لتوجيه الطلبات إلى الخدم الجاهزين للعمل . بنظرة أولى ، تظهر هذه المهمة وكأنها في عتق قنينة . ولكن ، يُسهل العمل بسبب كونه يتعلّق بمهمة « سلبية » : من الممكن تنفيذ عمليات قبول طلب الخدمة حسب قرينة التنفيذ للطالبين ، وعمليات قبول الطلب من الخدم .

غياب التعيين الأوتوماتيكي
 من الشائع أن تحتاج وحدة ترغيب بالإتصال والتبادل إلى التعريف عنها . الدواء
 المُسَكَّن لذلك (المستعمل في 8.7.6) يقوم ، كما يدل المثال الثاني ، عل تعريف مدخل
 إضافي يؤمن « تعמיד » المهمة . هذه الأخيرة تعرف إسمها ، يضاف إلى ذلك البلوغ الذي
 يسمح بتعيينها ، عندما يتم تحقيق التعמיד .
 مثال

declare

```

type T_TACHE ;
type A_TACHE is access T_TACHE ;

TACHE : A_TACHE ;

task type T_TACHE is
  entry BAPTEME (T : in A_TACHE) ;
  ...
end T_TACHE ;

task body T_TACHE is
  MOL_MEME : A_TACHE ;
begin
  accept BAPTEME (T : A_TACHE) do
    MOL_MEME := T ;
  end BAPTEME ;
  ...
  AUTRE_TACHE.ENTREE (MOL_MEME) ;
  ...
end T_TACHE ;

function NOUVELLE_TACHE return A_TACHE is
  N : A_TACHE := new T_TACHE ;
begin
  N.BAPTEME (N) ;
  return N ;
end NOUVELLE_TACHE ;

...
begin
  TACHE := NOUVELLE_TACHE ; -- إنشاء وتعמיד
  ...
end ;

```

ـ المهمة تُسَجَّل إسمها
 ـ المهمة AUTRE-TACHE تعرف من يُنادي مدخلها .

ملاحظة : كي تمتنع عن نسيان التعמיד ، يكفي إدخال T-TACHE في رزمة لا
 تُرسل سوى نوع خاص A-TACHE ، الدالة NOUVELLE-TACHE هي عبارة عن

إجراء حسياً يوجد مداخل لها غير التعميد BAPTEME .

و - الوقت

في كل نظام آدا ، يوجد مفهومان مستقلان للوقت ولكنهما متممات لبعضهما :
الترزامة للوقت المطلق ، والتعير عن فسخة الوقت التي نجدها في طلبات التعليق ، كلاب
الحماية ، ومفهوم الاستعمال . كل طلب للترزامة يجعل النتيجة دقيقة : الساعة . وعلى
العكس ، فإن مفاهيم الفسحات الزمنية هي غير دقيقة ، فهكذا : - موعد مع مرة من N
ثانية لا يعني أبداً إن الموعد المحتمل هو مأخوذ في الحسبان قبل N ثانية .
- موعد مستعمل ومباشر لا يعني أبداً أن الموعد سيؤخذ بالحسبان في نفس لحظة طلبه .

التعليمة delay هي صالحة للاستعمال في حالتين : لتأخير المهمة أو لتسليم طلب
الحماية (chien de garde) بشكل يتحدد فيه مدة إنتظار أحد الشريكين في الموعد . بشكل
عام ، هكذا كلاب حماية تسمح باكتشاف خطأ في البرمجة (إغلاق داخلي) ، التحميل
الزائد للخادم ، عطل في العتاد (خط الاتصال مثلاً) ، الخ . التشغيل الجيد للكلاب
الحماية يتطلب وضع بروتوكول تفسير للموعد الذي يحفظ تماسك حالات الشركاء . هذا
البروتوكول يجب أن يكون مستوفاً بواسطة أوالية الاتصال بين المهام ، مما يجعل من الصعب
اكتشاف الأخطاء التي تؤثر على هذه الأوالية .

كما هو معروف ، فإن اللغة ليست مستعملة للأعمال التطبيقية في التحكم بالعمليات
حيث يوجد إلزام دقيق جداً في مدة الجواب . وعلى العكس ، لا شيء يمنع أن تكون بعض
أعمال التشغيل متكيفة مع هذه الفضة من التطبيقات ، شريطة أن تتوقع خوارزميات
تخصيص الموارد إمكانيات الإسترداد حسب الأولويات النسبية للمهام (وبالأخص عند
عمليات الانقطاع) .

8.8.3 الأخطار

سنعالج هنا الأخطار الناتجة عن الاستعمال السيء لبعض الإنشاءات .

أ - الكيفي والعشوائي

عملية إختيار أحد الفروع القابلة للاجتياز من تعليمة Select تتم بشكل « كيفي » .
الاختيار العشوائي قد يُسبب غياباً في الموارد (أي الاتعداد المنتظم عن أحد الفروع) .
الكيفي هو موجه لترك الحرية الكاملة لكتاب برامج التعريف . وعلى عائق المبرمج تقع
مهمة تأمين غياب الموارد حسب الحوارزم .

ولكن ، المهام المصرح عنها في نفس القسم الوصفي تصبح « فُعاللة » بترتيب
كيفي . عملية « التفعيل » تقوم على تصميم أقسام وصفية لأجسام المهام ، ويجب على
المبرمج أن يتفادى المردود السيء الناتج عن هذه الأقسام الوصفية . مثلاً ، الجسم المعتمد

لنوع مهمة يمكن أن يُصرَّح عن جفول بأبعاد محسوبة عند كل عملية تصريح عن الموضوع مهمة ، والنتيجة تُهَد بأن لا تكون دائماً قابلة للتقدير .

ب- الفروع المدافع عنها والأوامر المدافع عنها

مع إن البناء Select يؤدي إلى إنشاء أمر مدافع عنه ، فليس من الممكن منحه نفس الخصائص المنطقية الخاصة بالأمر عندما يؤخذ أحد المواعيد في الحسبان ، فالمدافع المناسب يمكن أن لا يتم التدقيق به . إضافة لذلك ، فاختيار الكلمة المحجوزة When يمكن أن يؤدي إلى خطأ .

مثال

```
select
  when CALENDAR.CLOCK = MIDI => ...
end select;
```

فضلاً عن ذلك فإن تقييم المدافعين ليس ذرياً . ولا ينصح بمراجعة المتحولات المُقسمة من المدافعين (أي المُشار إليها بواسطة مهام أخرى) .

مثال

```
select
  when G >= 0 => accept E1 ;
  or when G < 0  => accept E2 ;
end select ;
```

- قد يؤدي إلى إثارة حالة إستثنائية هي SELECT-ERROR

وعلى عكس كل إنتظار ، فإن الحالة الإستثنائية SELECT-ERROR يمكن أن يتم إطلاقها إذا جرى تعديل المتحولة i بواسطة مهمة أخرى وخلال تقييم المدافعين . تعريف اللغة أدا يترك الحرية الكاملة للعاملين في اختيار خوارزميات إنشاء المدافعين واختيار المواعيد . هذه الحرية هي غالباً علودة بسبب وجوب حساب وتقييم كل مدافع .

حالة خاصة : الخاصية COUNT

الخاصية 'COUNT' هي حالة خاصة عبارة عن متحولة مقسومة .

مثال :

```
select
  when E'COUNT >= 2 => accept E ; accept E ;
or
  ...
end select ;
```

عند تنفيذ التعليمات الأولى accept ، من الممكن ألا يوجد سوى طلب واحد لموعد ، والطلبات الأخرى للمواعيد جرى سحبها بسبب إنتهاء المدة المحددة في تعليمات النداء . من الممكن بشكل عام منع إستعمال النداءات المحدودة في الوقت بوضع المهام داخل رزم تنقل لإجراء معين لكل مدخل .

ج - التوقيف الداخلي

بما إن المهام ليست معرّفة بشكل ساكن ، فمن غير الممكن إكتشاف أخطار التوقيف الداخلي عند التصريف . هذه الأخطار هي نوعان : التوقيف الداخلي « الكلاسيكي » لواحدة أو عدة مهام وبالتالي لشلل النظام الثانوي ، أي لمجموعة من المهام المرتبطة بعلاقات « تبعية » (أنظر 8.4.1) .

التوقيف الداخلي (interflocage) الصافي لواحدة أو عدة مهام

- يمكن للمهمة أن تستدعي : وهي ستبقى متوقفة لمدة طويلة . هذه الحالة البسيطة ليست دائماً قابلة للإكتشاف بشكل ساكن .
- إنتظار متبادل لعدة مهام : هذه الحالة تتج بشكل عام عند النداءات المتقاطعة .
- برجة سيطرة المهام الزامنة أو للدفاعيين في تعليمات إنتظار إنتقالي .

شلل مجموعة من المهام التوابعية

- إن وجود مهمة حيث شروط الإنتهاء ليست دائماً كافية يمكن أن يمنع تطور الوحدة التي تتعلق بها المهمة .

- البرجة السيطرة للدفاعي التعليمات Select ، وبالتحديد مدافعي عملية الإنتهاء التي تمتاز بفائدة بعيدة عن كونها حتمية ، يمكن أن تمنع أخذ شروط إنتهاء المهمة بعين الاعتبار .

د - تماسك المعطيات

تبلغ المهام بشكل حرّ جميع المواضيع المرئية . ويعود للمبرمج مهمة تأمين التماسك بين المعطيات عندما تكون مبلوغة بالتوازي . مثلاً ، وبهدف الحصول على صيغة مثل ، نفترض مصفّات أدا عدم وجود تنازع على البلوغ إلى المتحولات المشتركة : كل مهمة يمكن أن تحفظ نسخة عنها في مرافقها . هذه الحرية المتروكة للعاملين ترفع قليلاً كل دلالة عن مفهوم ملئ المتحولات المشتركة .

يمكن أن يقوم المبرمج بزيادة زخم الإستيفاء اليومي للمتحويلات المشتركة بواسطة نداء نموذج للإجراء الأصلي SHARED-VARIABLE-UP . (أنظر 8.3) . جميع مشاكل الإستيفاء اليومي لا يمكن أن تحل بهذه الطريقة ، مثلاً ، عندما لا تحفظ قيمة معينة على وحدة الآلة ، فالبلوغ المتزامن يمكن أن يعطي نتائج زائفة . ويجب على المبرمج أن يقوم بالزمانة الضرورية .

لنضف أنه في وجود علة مهام ، فلا الاجراء الأصلي
[9.11] يمكنه تصحيح الأثار المترتبة عن الحرية في الحصول على صيغة مثل والتروكة
للمصرفات .

أفليس من العدل ترك الحرية للمبرمج للعناية بالإشارة إلى المصرفات عن المتحولات
الواجب أن يتم نسخها ؟

8.8.4 التناقض المرفوع بواسطة [MRA] على مشكلة الحالات الإستثنائية في وجود المهام
(الحالات الشاذة جرت معالجتها بشكل مفصل في الفصل التاسع)

بعض التناقض المتعلق بمفهوم الانتهاء ، يظهر بين [MR9.4] ، ومعالج المهام
و [MR 11.4.1] تعالج الحالات الشاذة .

يُكتب في [MR 11.4.1] ، أننا لا نترك وحدة من البرنامج طالما إن المهام التابعة لم
تنته بعد . أكثر من ذلك نُحدّد إن هذا يتضمن الحالة التي نصل فيها إلى نهاية جسم الوحدة
عند وجود حالة إستثنائية غير مسترجعة . وفي النهاية ، يُقال إن الإنتهاء الطبيعي لمهمة
معينة يحصل ، إما عندما تبلغ هذه المهمة التعليمية end لنهاية جسمها وتنتهي معها جميع
المهام التابعة لها ، وإما عندما يجري إختيار فرع إنتهاء مفتوح .

[MR 11.4.1] يدل على ما يجري عندما نلتقي حالة إستثنائية خلال تنفيذ جسم
المهمة أو جسم برنامج ثانوي ، فدرة ، رزمة أو مهمة . لناخذ ، مثلاً ، النقطة د من هذه
الفقرة :

« عندما يتم إطلاق حالة إستثنائية في متتالية تعليمات جسم المهمة الذي لا يحتوي
على معاود أو مراجع للحالة الإستثنائية ، فتنفيذ المهمة هو معلق ومتروك ، والمهمة
منتبهة » .

جرى إستعمال المصطلح معلق أو متروك بدون تعريف . [MRA] تشير إلى إن
المهمة تمر في الحالة المنجزة ، في إنتظار إنتهاء الحالات التابعة ، إنتشار حالة إستثنائية غير
مسترجعة لا يتم إلا عندما تكون جميع المهام المتعلقة بقدرة أو برنامج ثانوي قد إنتهت .
هذه النقطة ليست واضحة في [MR 11.4.1] .

حول معرفة الحالة المنجزة

أحد إنجازات [MRA] بالنسبة لـ [MR] هي التمييز بين الحالة المنجزة والحالة
المنتبهة . في الحالة الأولى ، لا يمكن للمهمة أن تستقبل نداءات على مداخلها ، وهذه
الأخيرة تقوم بإطلاق الحالة الإستثنائية TASKING-ERROR . وهي تتمتع بمهام تابعة

غير منتهية . وفي الحالة الثانية ، تكون المهمة والمهام التابعة لها كرفع أو كشعبة مية تُتابع المهمة وجودها وكأنها معنية كموضوع .

حسب [MR 4.8] ، فالمساحة المشغولة بواسطة موضوع منشأ بواسطة مُخصّص يمكن أن تُستعاد منذ أن يصبح الموضوع غير مبلوغ . وتُصيف [MRA] ، وهذا يبلو طبيعياً ، إن الإستعادة لا يمكن أن تحصل إذا كان الموضوع عبارة عن مهمة غير منتهية .

حول التعليمية Abort

تؤدي التعليمية Abort إلى إنهاء المهمة المعنية ، إضافة إلى جميع المهام التابعة لها بشكل إنتقالي . [MRA] تشير إلى أنه ، وبعد تنفيذ التعليمية Abort ، تقيد الخاصية T'COMPLETED وليس الخاصية T'TERMINATED القيمة TRUE [MRA] . لا تحتاج إلا إلى مزامنة قليلة . لن تصبح المهمة منتهية إلا داخلياً . وتصبح في الحالة التي لا تسمح لها بقبول النداءات على مداخلها .

حول أوالية Terminate

[MR] يقدم تفسيراً مركزياً لقواعد إنتهاء المهمة . وهو يذكر أن المهمة المنتظرة في تعليمية Select مع terminate يمكن أن تصبح منتهية عندما تصبح الوحدة T والمهام المتعلقة بهذه الأخيرة هي إما منتهية ، وإما موضوعة في الانتظار في تعليمية select مع Terminate .

فلنفترض إن هذه الوحدة T هي نفسها مهمة في الانتظار في تعليمية Select مع Terminate . فشرط إنتائها يتعلّق كما في السابق بحالة الوحدة التي تتعلّق فيها وبحالة المهام التابعة لهذه الأخيرة . وإذا لم يتحقّق هذا الشرط ، فالمهمة T يمكن أن تصبح فعالة بواسطة نداءات لواحدة من المداخل المرتبطة بفروع الموعد المفتوح ، والرغبة في نداء مداخل من مهمة تابعة لها . وبما إن هذه الأخيرة قد إنتهت ، فإن الحالة الإستثنائية -TASK ING- ERROR سيتم إطلاقها .

[MRA] يحل هذه المسألة باعتماد قاعدة التبعية الإنتقالية وبإدخال المفهوم top unit (الوحدة حيث المهمة تتعلّق بها إنتقالياً والتي بلغت الأمر end في نهاية جسمها) .

حول قواعد التبعية

في [MR] وفي [MRA] ، مكتوب إن كل مهمة يمكن أن تتعلّق برزمة من مكتبة ، بينما لا يمكن أن تتبع أو تتعلّق برزمة داخلية . هذا سيبدو وكأنه يشير إلى أن كل مهمة مُغلقة بداخل تعليمية select مع terminate ، وتابعة لرزمة من مكتبة ، يمكن أن تصبح منتهية عندما تبلغ الرزمة الأمر end في نهاية جسمها وتكون جميع المهام التابعة إما منتهية ، وإما مُغلقة بداخل تعليمية select مع terminate . وهذا قد لا يكون مرغوباً به .

العلاقة التبعية ستصبح عمولة إلى وحدة وهمية في محيط النظام ، الذي يتمتع بجملة حياة .
تعاود مدة حياة البرنامج المركزي .

حول الحالة الإستثنائية FAILURE

في Ada ، نحدد الحالات الإستثنائية في مفهوم متالي ؛ فلا يمكن أن تكون متشعبة
بين المهام . في [MR] ، كانت الحالة الإستثنائية المحددة بـ FAILURE مربوطة
بكل موضوع من المهمة ويمكن أن يتم إطلاقها في أي من الوحدات التي تتمتع ببلوغ إلى
موضوع المهمة .

باستعمال أكثر بساطة من التعليمة abort ، وسبب كونها كانت تعطي الإمكانية
للمهمة المتعلقة بتنفيذ مسترجع ، فالحالة الإستثنائية FAILURE كانت تبدو مفيدة لضبط
البرامج والبحث عن نقاط التوقيف الداخلي . واستعمالها يفرض اليوم كثيراً من المشاكل ،
مما يبرر إلغائها في [MRA] .

يجب أن تكون المهمة ، المغلقة بواسطة end في نهاية جسمها وذلك في إنتظار
إنهاء المهام التابعة ، قادرة على إستقبال الحالة الإستثنائية FAILURE ، وهذا ليس
لإعادة إطلاقها مع المهام التابعة لها . يمكن للمهمة ، المغلقة بواسطة end في نهاية
جسمها ، أن تصبح فعالة لتنفيذ مسترجع ، وحتى بالنسبة لاستقبال نداءات على
مداخلها . وجود الحالة الإستثنائية FAILURE كان يمنع التمييز الواضح للحالة
المنجزة . نداء أحد المداخل لم يكن يؤدي إلى إطلاق الحالة الإستثنائية
TASKING-ERROR إلا عندما تكون المهمة في حالة الإنتهاء . إضافة لذلك ، يبدو
، دون أن يكون ذلك واضحاً ، أنه بإمكاننا إطلاق الحالة الإستثنائية FAILURE لعدة
مرات على نفس المهمة المسترجع المعتمد لا يكون قد جرى تنفيذه لمرة واحدة .

البحث عن أولية تسمية صائبة للحالات الإستثنائية منذ التصريف كان يمنع
التصريح عن مُسترجع الحالة الشاذة FAILURE في برنامج ثانوي . هذا المنع يقوم
بإدخال حقيقي في حالات التوقف الداخلية ، في وجود المهام التابعة . سيبدو من الممكن
إسترجاع الحالة الإستثنائية FAILURE في الأمر others .

الإسم المعطى للحالة الإستثنائية يمكن أن يجعل المبرمج يصدق بأنه كان على حق
بانتظار أن المهمة التي تستقبل هكذا حالة إستثنائية توقف عملها لمدة طويلة . ولكن ، قد لا
تكون هي الحالة التي تكون المهمة فيها قد دخلت في فترة أو جرى استدعاؤها من برنامج
ثانوي . يمكن للحالة الإستثنائية أن تكون مسترجعة في الفترة أو في كلمة others في
برنامج ثانوي ، وإما غير مسترجعة وتؤدي إلى توقيف داخلي ، وتبقى الفترة أو يبقى
البرنامج الثانوي متوقفاً في إنتظار إنتهاء المهام التابعة .

8.9 ختام

تسمح لغة آدا بسهولة التعبير عن مشاكل التزامنة والاتصالات التي نلتقيها عادة في برمجة النظام . وبإدخالهم لمفاهيم الإنقطاع إلى لغة آدا ، أو مفاهيم الساعة ، الأولية ، وغير ذلك ، أراد المؤلفون أن يجعلوا من هذه اللغة لغة للبرمجة في الوقت الفعلي . من المناسب أن يكون الشخص حذراً لأن معنى هذه المفاهيم يتعلّق بالعاملين . والبرمجة في الوقت الفعلي ليست متأتية من اللغة فقط ، ولكنها أيضاً نتيجة منهجية معينة في العمل والبرمجة .

لا تسمح لغة آدا بالقيام بكل شيء . إنها لغة برمجة للنظام ، وليس للحساب المتوازي . لا تسمح أية أولية ، بشكل خاص ، بالتعبير عن توازن متزامن بشكل دقيق .

إذا لم تتوافق مفاهيم التزامنة والاتصالات بواسطة المواعيد مع جميع التركيبات المركزية والموزعة ، فالأمر ليس كذلك بالنسبة للاتصال بواسطة متحولات مقسمة . في غياب أية وسيلة للتكيف المحدّد للمهام في اللغة (أنظر 12) ، يضطر المستعمل الذي يصطدم بعمليات إلزام مختلفة لأن يأخذ التركيبية والبنية في الحسبان عند تصوره للبرنامج .

لا يوجد أولية حماية خاصة بالمهام في لغة آدا . الاستعمال المتصل للمهام والرزم يمكن أن يُحسّن مفهوم أمانة البرامج .

الفصل التاسع

الشواذ أو الإستثناء

9.1 تقديم لمشكلة الشواذ أو الإستثناء

الهمّ الحالي لمصممي المناهج هو في تقديم برامج فعالة وعاملة ، أي وبشكل بديهي ، برامج صحيحة بالنسبة لمواصفاتها ، ولكن إضافة لذلك ، مقاومة للأخطاء أو الحالات الشاذة أو الإستثنائية الناتجة عن المحيط حيث هي موضوعة . (مثلاً : حجم ذكرة غير كافي عند التخصيص) .

من ناحية أخرى فإمكان برنامج ثانوي أن ينتج حالات شاذة إستثنائية ؛ والأمثلة متعددة : القراءة من السجل ، إذا كان الموقع الجاري هو في نهاية السجل ؛ الدالة التي تعيد القيمة في قمة مكدمس ، عندما يكون المكدمس فارغاً ؛ حلّ نظام معادلات خطية ، عندما يكون المميز صفراً ، الخ . من الأخطاء الواجب أن نتفادها نذكر [Ployette 79 و Banâtre] .

- الحالات الشاذة الإستثنائية ، التي يُتوقع إكتشافها وتحديد موقعها ومعالجتها منذ تصور البرنامج .

- الأخطاء غير المسبقة ، والتي لم يتم توقعها ، هي غالباً عبارة عن أخطاء في التصور . معالجة أخطاء كهذه (تدعى أيضاً أخطاء باقية) تتطلب طريقة إستعادة منتظمة ، حتى يتم إعادة النظام إلى حالة صالحة لمتابعة التنفيذ .

مفهوم « الشواذ » يغطي هاتين الحالتين . وبشكل عام ، يمكن أن تطلق الحالة الإستثنائية E (raised) عند تنفيذ برنامج (ثانوي) P . هذا الإطلاق هو تفرع مباشر إلى متتالية معالجة تدعى مرجع E (handler) . إذا كان هذا الأخير معرّفاً بالنسبة لـ E . عندما ينتهي تنفيذ المرجع ، نحصل على عدة أنواع ممكنة من العودة . هذه هي المخططات الثلاثة الموصوفة في أغلب الكتب :

- 1 - مخطط « التصحيح » . هناك محاولة لمعاودة P مباشرة بعد نقطة إنطلاق الحالة الإستثنائية .
- 2 - مخطط « الإنتهاء » . يعتبر البرنامج (الثانوي) P متتياً ، ويعاد التحكم إلى فدرة تُغلف المرجع E .
- 3 - مخطط « محاولة جديدة » . مرجع الحالة الشاذة الإستثنائية E يدل على إن تأثيرات P ليست ذات دلالة ، وترقم القيم الداخلة إلى برنامج (الثانوي) P ويعيد إطلاق P أو أحد البرامج (الثانوية) المتناوبة P .

لن نُفصل هنا مختلف أليات الحالات الإستثنائية ومخططات التحكم . ويمكن أن نذكر هنا بعض المراجع عن هذا الموضوع : [Goodenough 75] [LEVIN 77] [HORning et al 74] [Randell 75] [Melliard- Smith, Randell 77] إضافة إلى لغات تُدخل أليات إكتشاف ومعالجة الحالات الإستثنائية : On condition : من 1 / PI. [Maclaren 77] ، CLU. Mesa .

9.2 مفاهيم آدا .

تحتوي آدا على أليات إستثناء تتبع المخطط (2) «terminaison» ، بشكل متوافق مع دفتر الشروط [Steelman 79] .

الحالة الإستثنائية هي عبارة عن حادثة غير طبيعية يمكن أن تحدث خلال تنفيذ البرنامج (أو التعليمية) . العملية التي تقول بأن الحالة الإستثنائية قد أطلقت هي عملية نادرة بحد ذاتها ؛ وهي عبارة عن إنقطاع المتتالية . والمعالجة المرتبطة بهذه الحالة الإستثنائية هي التي تعطيلها معناها ؛ وهذا قد يكون مثلاً حالة خاصة للقيمة ، أو خطأ في النداء في قيم متغيرات وسيطية ، أو محيط مغلووط ، الخ ...

9.2.1 التصريح عن الحالة الشاذة أو الإستثنائية

الحالة الإستثنائية هي عبارة عن معرف ، لا يتطلب أية متغيرات ، ويمكن أن يُنظر إليه ككاتبته من نوع مرقم وعُدد [MR 11.1] . وتحتوي على تكويد موحد تتم صياغته عند التصريف ، ويمكن للحالة الإستثنائية أو الشاذة أن تنتشر ديناميكياً خارج مدى التصريح (أنظر 9.2.4 ، مثلاً) .

مثال 1 :

التصريح عن الحالة الشاذة أو الإستثنائية

RESOLUTION - IMPOSSIBLE . exception;

9.2.2 مُسترجع الحالات الشاذة أو الإستثنائية

يمكن أن يظهر واحد أو عدة مُسترجعين للحالات الإستثنائية بعد الكلمة - مفتاح exception ، وفي نهاية وحدة ، أي :
- فِدرَة

- جسم برنامج ثانوي ، رُزمة أو مهمة .

عندما يتم إطلاق حالة إستثنائية عند تنفيذ الوحدة ، يجري قطع للتوالي الطبيعي من التعليمات ، والتحكم ينتقل إلى مسترجعي الحالات الإستثنائية في هذه الوحدة .

إذا جرى تعريف المُسترجع للحالة الإستثنائية المُنتقلة ، فيتم تنفيذه ، وإلا يتم تنفيذ المُسترجع المرتبط بـ Others إذا كان موجوداً ، وإلا في النهاية ، يجري انتشار الحالة الإستثنائية . تمتاز تعليمات المُسترجعين بنفس الحقوق في البلوغ كتعليمات الوحدة (متحوّلات مركزية ، متغيرات ، الخ) . لا نستطيع ، بواسطة تعليمة goto ، نقل التحكم بالتعليمات بشكل جلي من وحدة إلى مُسترجع ، ولا من مُسترجع لحالة إستثنائية إلى مسترجع آخر ، ولا من مسترجع إلى تعليمات الوحدة . أي العودة في الوحدة حيث الحالة جرى إطلاق الحالة الإستثنائية . في المسترجع ، لا نعرف سوى إسم الحالة الإستثنائية التي جرى إطلاقها ؛ لا يوجد هناك معلومات عن نقطة إنطلاق الحالة الإستثنائية ، وبشكل خاص بسبب غياب المتغيرات الوسيطة للحالات الإستثنائية .

مثال رقم 2

Declar

```
N : INTEGER ;
C1, C2, C3, V1, V2 : FLOAT ;
RESOLUTION_IMPOSSIBLE : exception ;
procedure RACINES (A, B, C : FLOAT ; R1, R2 : out FLOAT) ;
    AX2 + BX + C = 0 للمعادلة R2 و R1 للجذور
.. وتقوم بإطلاق RESOLUTION-IMPOSSIBLE إذا كان المُميز سلبياً ،
```

.. أو في حالة الغرض في القيمة ، أو القسمة على صفر .

```
begin
    GET (N) ;
    for I in 1 .. N loop
        begin GET (C1) ; GET (C2) ; GET (C3) ;
            PUT ("coefficients :") ;
            PUT (C1) ; PUT (":") ; PUT (C2) ; PUT (":") ; PUT (C3) ;
            RACINES (C1, C2, C3, V1, V2) ;
            NEW_LINE ;
            PUT ("racines :") ;
            PUT (V1) ; PUT (":") ; PUT (V2) ;
```

```

exception
  when RESOLUTION_IMPOSSIBLE =>
    PUT ("résolution impossible") ;
end ;
end loop ;
end ;

```

9.2.3 - التعليمة raise

وهي التعليمة التي تسمح بإطلاق الحالة الإستثنائية

مثال رقم 3

- تعريف جسم الإجراء RACINES ، حيث الحالة الإستثنائية المحلدة NUMERIC-ERROR يمكن أن تحدث مثلاً بـنتيجة القسمة على صفر .
- مثلاً بـنتيجة القسمة على صفر .

```

procedure RACINES (A, B, C : FLOAT ; R1, R2 : out FLOAT) is
  D : FLOAT ;
begin D := B * B - (4.0 * A * C) ;
  if D < 0.0 then
    raise RESOLUTION_IMPOSSIBLE ;
  else R1 := (- B + SQRT (DELTA))/(2.0 * A) ;
    R2 := (- B - SQRT (DELTA))/(2.0 * A) ;
  end if ;
exception
  when NUMERIC_ERROR => raise RESOLUTION_IMPOSSIBLE ;
end RACINES ;

```

لا يمكن أن تظهر التعليمة raise بدون معرف عن حالة شاذة إستثنائية إلا في مُسترجع للحالة الإستثنائية التي تطلقها . وهي ليست ضرورية بشكل فعلي إلا في حالة الأثر traces ، وعندما نعمل الحالة الإستثنائية التي سببت في الخروج عن المسار ، ولإنتشار الواضح للحالة الإستثنائية .

مثال رقم 4

```

exception
  when others => ... -- مخرج traces
                  ... -- إغلاق السجلات
                  raise ;
end ;

```

9.2.4 - ربط المُسترجعين والحالات الإستثنائية

الربط بين الحالة الإستثنائية والمُسترجع هو ديناميكي ، أي أنه يتعلّق بترتيب النداء

في البرنامج ، وليس بالإدخال الساكن للتصاريح . يجب الإلتباه أيضاً للموقع الذي تنطلق منه الحالة الإستثنائية في الوحدة . النتيجة لن تكون بالضرورة هي نفسها حسباً إذا كان ذلك يتعلّق بالقسم الوصفي ، أو بقسم إسترجاع الحالات الإستثنائية . في الجدول 1 ، نعرض الأجوبة على حالة إستثنائية في حالة التوالي . الحالة المتعلقة بالمهام هي معالجة في 9.2.5 في الحالة التي تحتوي فيها الفكرة أو جسم برنامج - ثانوي أو رزمة من ربيدة مناهج على تصريحات عن المهام ، فإن الحالة الإستثنائية لا تنتشر إلا عندما تنتهي المهام التابعة ، أنظر 8.8.4 و [MR 9.4] .

إذا كانت الوحدة U هي البرنامج المركزي ، فإن الإنتشار يعني « تعليق » البرنامج .

مثال رقم S

من الممكن أن نستشير الأمثلة من [11.4.2, 11.4.1 MR] . وإليك غيرها [ME 12.5.2] .

```
package D is
  procedure A ;
  procedure B ;
end ;
procedure EN_DEHORS is
begin ...
  D.A ;                -- appel de A dans D
end ;
package body D is
  ERREUR : exception ;
  procedure A is        -- la procédure A peut déclencher ERREUR
  begin ...
    raise ERREUR ;
    ...
  end A ;
  procedure B is
  begin ...
    EN_DEHORS ;
    ...
    exception
    when ERREUR =>
```

- نداء الإجراء
الذي يمكن أن يؤدي إلى انتشار الحالة الإستثنائية
من A .
ERREUR
معالجة ERREUR

```
end B ;
end D ;
```

جسم الرزمة	الوحدة U هي جسم برنامج ثانوي قدرة لا تحتوي على تصريحات عن مهام	
تتشر الحالة الإستثنائية في الوحدة التي تغلف جسم الرزمة أو الرأس لوحة - ثانوية	الحالة الإستثنائية X تتشر في الوحدة التي تطلب البرنامج الثانوي في نقطة الطلب أو النداء	تطلق الحالة الإستثنائية X في قسم التصريحات أو في القسم مسترجع الحالات الشاشة للوحدة U أو الحالة الإستثنائية مستطلق في قسم التعليمات من الوحدة U وهذه الأخيرة لا تحتوي على مسترجع لـ X
تنفيذ مسترجع الحالة الإستثنائية X		
تطلق الحالة الإستثنائية X في قسم تعليمات الوحدة U وهذه الأخيرة تحتوي على مسترجع لـ X		

جدول 1

الإجراء EN-DEHORS يمكن أن يستقبل ERREUR بواسطة نداء D.A ، ولكن لا يمكنه إسترجاعه إلا بواسطة others . وهو ليس له رؤية ERREUR .
مثال على المعاليج المتداخل

```

exception
when DEBORDEMENT => ... [1]
    declare
    ...
    begin
    ... [2]
    exception
    when ERREUR => ... ;
    when others => ... ;
    end ;
    ... [1]

end ;
```


إنطلاق الحالة الإستثنائية في [1] يؤدي الى وقف تنفيذ المُسترجع والإنتشار حسب طبيعة الوحدة ؛ وعلى العكس ، فالحالة الإستثنائية في [2] هي مسترجعة في القدرة ، أما بواسطة ERREUR ، وإما بواسطة others . حسب قواعد الإنتشار ، فالبرنامج لا يمكن أن ينفذ على معالجات الحالة الإستثنائية .

9.2.5 حالة المهام

تحتوي المهمة ، كالوحدات الأخرى ، على قسم وصفي ، وقسم للتعليمات وقسم مراجعة للحالات الإستثنائية . وللمساعدة على الفهم ، سنسمي T للمهمة التي تنطلق فيها الحالة الشاذة الإستثنائية X ، و U هي المهمة أو الوحدة - الأم التي تتبع لها المهمة T [8.4.1] ، V هي المهمة التي قد تكون على موعد مع T .

- إذا جرى إطلاق X في قسم التصريح من T ، فتكوين T هو متروك والحالة الشاذة تنتشر إلى U ، وذلك في القسم الخاص بالتعليمات ، عندما تنتهي جميع المهام التابعة لـ T .

- إذا جرى إطلاق X في قسم التعليمات من T ، وخارج التعليمات accept ، وإذا كانت T تحتوي على مُعالج لـ X ، فسيكون هناك تنفيذ للمُعالج .

- إذا جرى إطلاق X في قسم التعليمات من T ، وخارج التعليمات accept ، وإذا لم تحتو T على مُعالج لـ X ، فإن الحالة الإستثنائية لن تنتشر والمهمة T هي منجزة .

- إذا جرى إطلاق X في قسم مُراجعة الإستثناء ، وخارج التعليمات accept ، فإن المهمة هي منجزة والحالة الإستثنائية لا تنتشر .

- إذا كانت المهمة V المنادية هي على موعد مع T وإذا جرى إطلاق X في التعليمات accept E من T ، فهذه التعليمات مشترك ، والحالة الإستثنائية مستترة في نفس الوقت إلى نهاية التعليمات accept ، وإلى V في نقطة نداء E .

- إذا كانت المهمة V المنادية تحاول الدخول في موعد مع T . وإذا كانت T هي مُنجزة مع قبول للموعد ، فسيجري إطلاق الحالة الإستثنائية TASKING-ERROR في V وذلك في نقطة النداء . نفس الشيء سيحصل إذا انتهت T بشكل غير طبيعي خلال الموعد . (أنظر 8.4.4) .

- إذا انتهت المهمة المنادية V بشكل غير طبيعي خلال إنتظار الموعد مع T ، فسيجري سحبها من لائحة الدخول ؛ وإذا كان الموعد جارياً وقد حصل ، فستنتهي بشكل طبيعي في T .

في تلك الحالات ، فإن المهمة المناداة غير مُتغيرة .

9.2.6 الحالات الإستثنائية المحددة مسبقاً

يوجد 5 حالات إستثنائية محددة مسبقاً ؛ ويجري إطلاقها بشكل طبيعي عندما لا يمكن متابعة التنفيذ بشكل طبيعي (مثلاً : القسمة على صفر ، ...) . يوجد إمكانية للإشارة إلى المصروف بعدم توليد الكود المناسب لفحص بعض الحالات الإستثنائية ، أو لأحدى الحالات الخاصة من هذه الحالة الشاذة . مثلاً ، الحالة الشاذة الإستثنائية NUMERIC-ERROR يتم إطلاقها عندما تؤدي الحسابات الرقمية إلى فيضان في القيمة أو عند القسمة على صفر . من الممكن أن نقوم بإلغاء إنتقائي لإحدى حالات الفحص هذه . وهذا يتم بواسطة : SUPPRESS . ومن البديهي ، إذا إنطلقت الحالة الإستثنائية بينما تكون إمكانية إطلاق الإستثناء قد جرى إلغاؤها ، فنتيجة البرنامج لا يمكن التكهّن بها .

سنعطي في الجدول 2 الشواذات أو الإستثناءات المحددة والتدقيقات التي يقوم بها المصروف .

مثال رقم 6

إلغاء الفحوصات الداخلة في إكتشاف CONSTRAINT-ERROR يمكن أن يُجملد بمواضيع من نوع معين أو أيضاً بموضوع خاص .

type TABLE is array (1 .. 50) of INTEGER ;

- هذا الأمر يلغي الفحوصات حول صلاحية المؤشر في عناصر الجداول من نوع : TABLE . من الممكن أيضاً أن نقوم بإلغاء الفحوصات على موضوع خاص : TAB:TABLE ؛ في غياب الأمر السابق ، يمكن أن نكتب :

pragma SUPPRESS (INDEX_CHECK, ON => TABLE) ;

عما يؤدي إلى إلغاء الفحوصات للدليل مربوط بـ TAB

9.3 التنظيم

9.3.1 الدقة التي يتم إدخالها بواسطة [MRA]

هناك عدد من الأسئلة المفروضة عند قراءة [MR] ، التي نحصل على الإجابة عنها في [MRA] .

إعادة التصريح عن الإستثناء

يقال إن تعريف الإستثناء يتم إدخاله عند التصريح عنه وبشكل مساكين [MR] 11.1 . هكذا ، فإذا نظرنا إلى المثال 7 ، فالمصروف لا يُنشئ سوى الحالة الإستثنائية : I ، التي تنتشر على طول سلسلة النداءات المتتالية لـ P ، مع إن التصريح هو داخل الإجراء .

إستثناءات محددة	حالة خاصة في التدقيق	
CONSTRAINT_ERROR في جميع الحالات التي يوجد فيها إجبار من نوع مخالف [MR 9.3.3] . الأفعال التي تنبش فيها هذه الحالة الإستثنائية هي المذكورة في [MR 11.1]	ACCESS_CHECK DISCRIMINANT_CHECK INDEX_CHECK LENGTH_CHECK RANGE_CHECK	يتحقق من أن المؤشر المستعمل للبلوغ ليس صفراً يتحقق من الإجبارات بالنسبة لقيم المميز . يتحقق من أن قيمة الإشارة هي متكيفة مع الحدود من نوع دليل . يتحقق من أن عدد المركبات يعادل القيمة المحددة بواسطة النوع دليل [MR 4.3.2] . يتحقق من أن قيمة توافق التحديد في نوعها ، وفي حالات التحديدات الإجبار في الأنواع - الثانوية
NUMERIC_ERROR عدم إمكانية متابعة الحساب الرقمي	DIVISION_CHECK OVERFLOW_CHECK	يتحقق من أن المئات الثاني للقسمة ليس صفراً . يتحقق من حالات الفيضان في القيم
SELECT_ERROR لا يوجد قسم else بينا جميع فروع التلمية Select هي مغلقة .		
SOME_ERROR لا يوجد مكان في الذاكرة أو خطأ في ترتيب التصميمات عند تنفيذ أي حالة متوقعة بواسطة مصرف (متحولة غير مُعدّة) .	STORAGE_CHECK ELABORATION_CHECK	يتحقق من أن المكان في الذاكرة هو كاف للتخصيص . يتحقق من أن البرنامج الثانوي لم يطلب قبل تصميم جسمه
TASKING_ERROR إستثناء قد يحدث في إتصال بين المهام .		

مثال رقم 7

```

procedure P (I : in INTEGER) ;
  E : exception ;
  SAUVEGARDE : INTEGER ;
begin
  SAUVEGARDE := T(I) ; - تخزين قيمة متحولة عامة
  ... - E لا يمكن إطلاق
  if ... then raise E ;
  end if ;
  P (I + 1) ; - نداء متتال لـ P ، يمكن أن يساعد في إنتشار E .
  ...
exception
  when E => - يعيد استرجاع الإستثناء المطلق بواسطة raise أو المنتشر
               بواسطة النداء الداخلي لـ P
    T(I) := SAUVEGARDE ;
    raise E ;
end P ;

```

عند الخروج من النداء الأول لـ P ، الحالة الإستثنائية لا يمكن إستعادتها إلا في إختيار **when others** .

وفي حالة شبيهة ، حيث إعادة التصريح هي في مفهوم ساكن ، يوجد إنشاء لحالة إستثنائية جديدة تغطي على الأولى :

مثال رقم 8

```

declare
  E : exception ; - تصريح (1)
  procedure P ...
  ...
  raise E ; - إطلاق الحالة الإستثنائية المصرح عنها في (1)

end P ;
begin
  declare
    E : exception ; - تصريح فارغ (2) صالح ، مصادلة إستثناء جديد

  begin
    P ... ; - إنتشار عتمل لـ E مصرح عنه في (1)
    raise E ;
  exception
    when E => ... P - مسترجع E مصرح عنه في (2) ، ولكن لا يوجد E منتشر بواسطة P

exception
  when E => ... - مسترجع E مصرح عنه في (1)
end ;

```

إذا جرى التصريح عن الإستثناء في القسم المرئي للرمزة الأصلية (أنظر الفصل 10)، فهو يأخذ قيمة مختلفة في كل مثال مولد .

مثال رقم 9

```
generic
package P is
  E : exception
  procedure F ( ... ) ;      -- peut déclencher E
end P ;
package body P is
  ...
end P ;

with P ;
...
declare
  package P1 is new P ;
  package P2 is new P ;
begin
  P1.F ( ... ) ;             -- appel de F de P1 qui peut
                              -- déclencher P1.E
exception
  when P1.E => ... ;
  when P2.E => ... ;
end ;
```

- يمكن إطلاق E

فداه F من P1 الذي يمكنه إطلاق P1.E

إنهاء المهام ومعالجة الإستثناءات المرتبطة بالمهام

[MRA] يعرف جميع هذه المهام أفضل من [MR] ، حيث كانت الشروحات في [MR 11] في تناقض مع [MR 9] ، أنظر 8.8.4 . إلغاء الخاصية FAILURE يضاف بشكل كبير صعوبة أليات الإستثناء في Ada ، حتى ولو أنه في بعض الأحيان ، يؤدي إلى فقدان في قوة التعبير عنها .

9.3.2 - عدم كفاية أليات الإستثناءات

من المؤلف عدم كفاية أليات الإستثناءات :

- لا يوجد متغيرات للحالات الإستثنائية ، إطلاق الحالة الإستثنائية لا يمكن أن ينقل معلومات للمسترجع . وهذا قد يفرض مشاكل حول موضوع المدى بالنسبة للمواضيع المنقولة إلى المتغيرات ، والحل قد يكمن في عدم قبول إلا المتغيرات بالصيغة in .

- لا يوجد خاصيات تسمح للمسترجع بمعرفة ماذا جرى ، مثلاً عنوان إطلاق الحالة الشاذة أو إمكانية بلوغ مكتمل التنفيذ . . .
- لا يوجد إستثناء لإكتشاف غياب التصغير والإعدادات لمتحولة .
- لا يوجد إثبات كما في [GREEN] .
- الإستثناء المحدّد مسبقاً SOME-ERROR هو عبارة عن طريقة لاهمال النتائج .
- إذا كانت الأوالية سهلة في حالة البرامج المتتالية ، فهو معقد بالنسبة للمهام ، ونتيجة التنفيذ أزو إطلاق الحالات الإستثنائية - على الأقل عندما لا تكون مركزية - ليست دائماً طريقة في البرمجة ، في حالة المهام . المفهوم الأكثر فائدة هو في تمييز العلاقة utilisation d'une ressource بدلاً من علاقة المفاعلة أو علاقة النداء في إنتشار الحالات الإستثنائية : هذا هو المخطط المعروف من قبل LEVIN . وإذا لم تحل جميع هذه المشاكل في هذا العرض الأخير ، فإن الحلّ الذي تقدمه آدا قد يكون الأفضل .

3.3.9 النقاط الإيجابية

النقطة الأولى الإيجابية هي وجود أوالية الحالات الإستثنائية والتي هي عبارة عن وسيلة مفيدة لإنشاء وفهم البرامج ، بشرط أن تكون هذه الوسيلة مفهومة وواضحة بشكل جيد من قبل من يرغب باستعمالها .

عملية تكوين البرامج بشكل تركيبي يمكن أن تصبح أفضل إذا لم تكن مزودين بهذه الأوالية ، لأننا في هذه الحالة لن نكون ملزمين ، للحصول على نفس البرامج ، بإجراء فحوصات إضافية . مثلاً ، لنفترض أننا نرغب بالقيام بسلسلة متكررة من المعالجات D_1, D_2, \dots, D_n ، ولكن حيث نداء D_{i+1} مشروط بحسن أداء D_i . الطريقة العادية تقوم على وضع متغير عودة إضافي ، B ، وهو صحيح إذا كان D_i سعى التنفيذ .

مثال رقم 10

```

with P_EXT ;                               رزمة خارجة لعمليات الإجراء D1
use P_EXT ;
procédure CONTROLE_TRAITEMENT is
  B : BOOLEAN := FALSE ;
  NON_TERMINE : BOOLEAN := TRUE ;
begin
  <<RETOUR>>
  D1 ( . . . , B ) ;
  if B then goto CAS_D_ERREUR ; end if ;
  D2 ( . . . , B ) ;
  if B then goto CAS_D_ERREUR ; end if ;
  . . .
  if NON_TERMINE then goto RETOUR ;
  else goto FIN ;
end if ;

```

<<CAS_D_ERREUR>>

- متتالية استرجاع الخطأ
- تركيز للبدء بمعالجة جديدة

```
B := FALSE ;  
if NON_TERMINE then  
  goto RETOUR ;  
end if ;  
<<FIN>> null ;  
end CONTROLE_TRAITEMENT ;
```

تركيبية التحكم هي ممزوجة بعدد من عمليات الفحص والتفرع النوعي للتركيبية «spaghetti» الخارجة من بياني السياق . إستعمال الحالات الإستثنائية يسمح بتركيبية مرئية للمخوارزم .

```
with P_EXT ;  
use P_EXT ;  
procedure CONTROLE_TRAITEMENT is  
  NON_TERMINE : BOOLEAN := TRUE ;  
begin  
  while NON_TERMINE loop  
    begin D1 (. . .) ; -- enchainement séquentiel des traitements  
      D2 (. . .) ;  
      ...  
    exception  
      when CAS_D_ERREUR =>  
  
    end loop ;  
end CONTROLE_TRAITEMENT ;
```

- إستثناء منطلق بواسطة D₁
- سلسلة إستعادة الخطأ
- تركيز لاعادة البدء بالمعالجة

على هذا المخطط بإمكاننا حتى إجراء حالات إستثنائية وعمليات استرجاع مختلفة حسب الإجراءات ، من الممكن أيضاً أن تقوم بأعمال إعادة جزئية للمتتالية D₁ ، بواسطة تركيب أكثر دقة للفردات .

- عدم الكفاية المشار إليه في 9.3.2 يمكن أن يجد حلولاً أقل أو أكثر تناسباً مع الصيغة الشكلية التي تعرضها Ada . مثلاً ، من الممكن إستبدال غياب المتغيرات بالإستثناءات باستعمال المحولات العامة . التعليمة : «assert C» التي يمكن أن تقوم بإطلاق IN-LINE ASSERTION-ERREUR ، مثلاً ، يمكن أن تستبدل بواسطة النداء
للإجراء :

```

procedure ASSERT (C : BOOLEAN) is
begin
  if not C then
    raise ASSERTION_ERREUR ;
  end if ;
end ASSERT ;

```

- من الممكن تقليد المخطط « محاولة جديدة » (أنظر 9.1) لقاء بعض الصعوبة في البرمجة ومع أوالية منهجية لترميم المتحولات . النموذج الاسامي سيكون :

```

<< REPRISE >>
begin
  ...
  if ... then raise E ; end if ;
  ...
exception
  when E => RESTAURATION ;
             MODIFICATION ;
             goto REPRISE ;
end ;

```

- نسبة للصفة البسيطة ، فإن عملية تنفيذ أوالية الإستثناء يجب أن تكون فعالة .
 - بالنسبة للأنظمة المعقدة ، فمن المهم أن نقدر على التحكم بانتشار الحالات الإستثنائية :
 إذا قام أحد البرامج بإطلاق الحالة الإستثنائية المهملة والتي لم تتم إستعادتها ، فيجب أن لا تنبت هذه الحالة الإستثنائية في النظام دون أن يكون هناك إمكانية في الحماية . في لغة آدا ، الحل يكمن بإغلاق النداءات للبرامج الثانوية والمهام في فدرات تستطيع وقف جميع الإستثناءات ذات المستويات الأقل بواسطة others ، والتي ترقم حالة جديدة صالحة لباقي التنفيذ . التحكم هو تراثي وهو يسمح بوضع مناهج مقاومة للأخطاء .

الفصل العاشر

الشمولية (النوعية) GENERICITE

10.1 مدخل.

10.1.1 أهداف الشمولية

الشمولية في Ada هي وسيلة فعالة لتثبيت البرامج الثانوية والرزم كمتغيرات بسيطة . نصّرُح عن وحدة شاملة نوعية عندما نحتاج إلى عائلة أو مجموعة من هذه الوحدات ، كل نموذج لا يختلف عن الآخر إلا ببعض المميزات ، حيث المعطيات ليست ضرورية في تعريف الوحدة . وهذا يؤدي إلى تضادي إعادة كتابة التطبيقات المختلفة لوحداث البرمجة المتشابهة . وهذه بعض الأمثلة : مجموعة من التصريحات المثبتة بواسطة ثوابت (مثلاً : جداول تُثبّت فيها حدود العملية الخاصة) ؛ أنواع مجردة محدّدة في رزم ، ومثبتة بواسطة أنواع (مثلاً : نوع مكلمس مجهز بدوال «empile» ، «depile» النخ . حيث نوع العناصر هو وسيط) ؛ برامج ثانوية مثبتة بواسطة أنواع وأدوال (مثلاً : فرز أحد الجداول حيث نوع العناصر وعلاقات الترتيب هي إلزامية) .

10.1.2 حالة العمل

خلال مدة طويلة في اللغات ، الطريقة المتبعة للحصول على برامج (إجراءات) شاملة ونوعية ، أي نماذج برامج قابلة للتكيّف مع حالات عديدة ، فإن الرجوع إلى المعالج الأولي (preprocesseur) . من خلال نصّ أولي ، هذا الأخير كان يقوم بأعمال تبديل للنص لتوليد كل نموذج (مثلاً : ماكرو معالج PL/I) . هذا النوع من المعالجة يفرض عادة معرفة اللغة الملحقه - لغة التحكم بالمعالج الأولي - وهو لم يكن يسمح بتأمين تصحيح النص الأولي . هكذا ، فتمنّج البرنامج لم يكن قابلاً للتدقيق ، لأن صلاحية النموذج تتعلق بالنصوص التي كانت تستبدل الأقسام الشكلية حالياً ، يوجد تياران في دراسة الشمولية : الأول مرتبط بطريقة تعريف الأنواع المجردة ، ويقوم على تعريف الأنواع المجردة المثبتة بواسطة أنواع (مثلاً : مكلمس حيث العنصر هي من نوع t شكلي ، وهو عبارة عن نموذج لنوع المكادس الصحيحة ، مكلمس أعداد حقيقية ، النخ) . الأعمال

الرئيسية حول الموضوع هي اللغات CLU ، Alghard ، ومن جهة أخرى [Burstall, Go-]
 [Thatcher et al. 78] و [Kanda 78] .

التيار الثاني خرج من الأعمال النظرية لـ D. Scott [74 Scott] إلى الحقل الرياضي حيث المواضيع ، الدوال والأنواع هي قيم متنوعة . من الممكن إذا تثبيت البرامج بواسطة هذه القيم . وهذا ما نسميه متعدد الأشكال (شكلية) Polymorphisme [Lehmann.]
 [Syth 77] [Milner 77] [Donahue 79] ، أنظر أيضاً للدراسة [80 Boute] واللغات ML [79 Gordon et al.] و Bussell . هذا المفهوم الثاني في شموليته ، ينشئ من الصعوبات التقنية في عمل اللغة ، مشكلة المحيط المرتبطة بالإجراءات الناتجة عن الدوال ، مشكلة التعريف وتعادل الأنواع ، الخ . الشمولية في Adit تركّز على المفهوم الأول وكانت مقيدة بشكل إرادي بالرمز والبرامج الثانوية . نماذج الوحدة الشاملة هي معروفة بشكل كامل ساكن بواسطة أوالية التوليد الجلية . سنعود إلى هذا الاختيار في الفقرة 10.4.3 .

10.2 الشروحات

10.2.1 عموميات

الوحدات التي يمكن أن يصرّح عنها وكأنها شاملة في لغة آدا هي :
 - البرامج - الثانوية (الإجراءات والدوال)
 - الرزم -

يتألف رأس الوحدة الشاملة من كلمة - مفتاح (Keyword, mot-clé): متبوعة بمتغيرات وسيطية شاملة . ولا يمكن أن تكون متبوعة إلا بمواصفة برنامج ثانوي أو رزمة (10.2.3) . جسم البرنامج الثانوي أو الرزمة الشاملة يمكن أن يستعمل كمتغيرات وسيطية للمواصفة ، ولكن الرأس لا يجب أن يتكرّر .

نموذج الوحدة الشاملة يمكن أن يتولّد بواسطة قيم خاصة لمتغيرات وسيطية شاملة بمساعدة تعليمة التوليد [12.3 MR] - كلمة مفتاح new .

مثال رقم 1

- مواصفة الدالة الشاملة SQUARING

generic

type ITEM is private ;

with function "o" (U, V : ITEM) return ITEM is <> ;

- المتغير الوسيط الأول هو نوع

function SQUARING (X : ITEM) return ITEM ;

- Définition du corps de SQUARING

```

function SQUARING (X : ITEM) return ITEM is
begin
  return X * X;
end SQUARING;

```

- * : هو مؤثر شكلي .

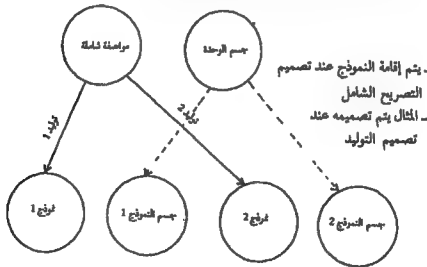
- إنشاء النموذج حيث المتغيرات الوسيطة الفعلية هي النوع MATRIX والدالة عمل للمصفوفات .
MATRIX-PRODUCT

10.2.2 قاعدة تصميم التصريجات الشاملة

التشكيل المستعمل في موضوع الشمولية هو التالي :

- تصميم التصريح الشمولي يؤدي إلى إقامة نموذج للوحدة المصروفة .
- جسم الوحدة الشمولية هو أيضاً عن نموذج لجميع أجسام النماذج المستقبلية .
- المعرفة التي تظهر في الوحدة الشمولية هي إما محددة مركزياً ، وإما عبارة عن متغيرات وسيطة ، وإما محددة بشكل عام . هذه الأخيرة هي مربوطة بالتعريف الذي يحيط بالتصريح الشامل .

- عند توليد النموذج ، يتم إنشاء نسخة عن الوحدة الأساسية ، مع وصل مختلف المتغيرات الشكلية والمتغيرات الوسيطة الفعلية (إذا كانت القواعد كافية أنظر 10.2.4) ، كذلك بالنسبة للجسم . إذا لم يتم إنشاء الجسم ، فإن التوليد هو غير صحيح بسبب وجود « بلوغ » لجسم غير مصمم أيضاً . نماذج المواصفة والأجسام هي أيضاً مصممة كتصريجات غير شاملة ، وربما مع تنفيذ أقسام الأعداد والتهيئة . قواعد وصل المتغيرات وتوليد النماذج ستكون محددة أيضاً في 10.2.5 .



خطط التصريجات الشاملة وإنشاء النماذج

- توضيح الوحدات بين المَعْرِفَات

decimre	
X : INTEGER := 3 ;	- تصريح 1 لـ X
generic package P is	- إنشاء مواصفة P
Y : INTEGER ;	
end P ;	- إنشاء جسم P
package body P is	
begin Y := 1 ; X := 2 * X ;	- X مرتبطة بالتصريح 1
end ;	
 begin	
declare X : INTEGER := 2 ;	- تصريح 2 عن X
package P1 is new P ;	- تصميم التصريح 2
package P2 is new P ;	P2 -
begin	
PUT (X) ;	- X داخل الفئدة = 2
end ;	
PUT (X) ;	- X من التصريح 1 = 12
...	
end ;	

عند النظر من الخارج ، نرى أنه لا يمكن إستعمال الوحدات الشاملة إلا كموديل للرمز أو للبرامج الثانوية ، وبالتالي ليس لها أي معنى إلا في التصريح عن مثال - أو في الأمر with إذا كانت عبارة عن وحدات من مكتبة . مثلاً ، من الخارج ، لا يمكن نداء إجراء شامل ، أو بلوغ مُركَّب من رزمة شاملة . في نفس المفهوم ، لا يمكننا تحميل زائد لمعرف برنامج ثانوي شامل (بينما نستطيع إجراء تحميل زائد لمعرفات الأمثلة) . على العكس ، في داخل جسم وحدة شاملة ، يمكن إستعمال معرف الوحدة مباشرة ، كما وكأنه يتعلّق برزمة أو برنامج ثانوي غير شامل (مثلاً ، في نداء متتالي - أنظر للمثال رقم 8) .

10.2.3 المتغيرات الشكلية الوسيطة للوحدات الشاملة

يمكن للوحدة الشاملة أن تمتاز بمتغيرات وسيطة هي :

- مواضيع أو قيم

- أنواع

- برامج ثانوية .

مثال رقم 3

generic	- وسيط عبارة عن قيمة
SIZE : NATURAL ;	- وسيط عبارة عن نوع
type ENUM is (<>) ;	
with function IMAGE (E : ENUM) return STRING ;	- وسيط عبارة عن دالة
function F (...) ... ;	- مواصفة الدالة الشاملة

معرفات المتغيرات الوسيطة الشاملة يمكن أن تستعمل بعد النقطة - الفاصلة التي تُنهي التصريح . نفس الشيء بالنسبة للعمليات أو الخاصيات المرتبطة بنوع شكلي (أنظر 10.2.3.2) . يبدأ مدى معرف الوحدة الشاملة بالكلمة - المفتاح generic ؛ ولكنه لا يمكن أن يُستعمل إلا بعد التصريح نفسه .

مثال رقم 4

```
generic
  SIZE : NATURAL ;
  HEIGHT : NATURAL := SIZE ;
  type INT is range <> ;
  ELEM : INT := INT*FIRST + 1 ;
procedure P ( ... ) ;
```

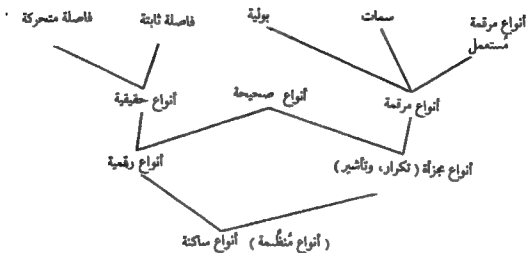
- تصريح مع قيمة بالغلط
- تصريح عن نوع صحيح

القيم الوسيطة

متغيرات وسيطة عبارة عن قيم تمرّ القيم في الصيغة in (بالغلط) أو في الصيغة out فقط [MR 12.1.1] . المتغيرات الوسيطة in يمكن أن تحتوي على قسم إعداد سيؤخذ كقيمة بالغلط إذا كان المتغير الوسيطي الفعلي المناسب غير معطى في توليد المثال . المتغير الوسيطي في الصيغة in لا يمكن أن يكون من نوع محدود . من الممكن أن نُدesh لعدم وجود الصيغة out كما في الإجراءات ؛ وفي الحقيقة ، فإن الصيغة in out هي مأخوذة في معنى يختلف عن المعنى الموجودة في الإجراءات : in out يعادل إعادة تسمية المتحولة عن توليد المثال (أنظر 10.2.5) . فلنذكر أنه في جسم وحدة شاملة ، فإن الاختيار ، التعريف عن النوع الصحيح والإلزام بالدقة كل هذا لا يمكن أن يتعلق بالمتغيرات الوسيطة الشاملة [MR 12.1] . وبشكل عام ، فإن المتغيرات الوسيطة الشاملة لا تعتبر كقيم ساكنة .

مواصفات المتغيرات الوسيطة الشكلية ونوع ، [MR 12.1.2] يتألف النوع الشكلي من معرف وربما أيضاً من قسم يُميّز (بدون قيمة بالغلط) ، وتعريف شكلي هو («generic-type-definition») . سنعطى أدناه بعض الجداول لفهم ميكانيكية مواصفة الأنواع الشكلية .

من الأنواع البسيطة (غير الخاصة) ، الجدول T1 يعيد التذكير بالمصطلح rapport ، فئات الأنواع التي تغطيها الكلمات .



T1 - فئات الأنواع البسيطة المخططة مسبقاً

من المهم أن نلاحظ إن مواصفة متغيرات الأنواع هي هنا لتحديد العمليات الدنيا التي نحتاج إليها في الوحدة الشاملة . من الممكن أن نوجز هذا بواسطة الجدول T2 .

في جميع الحالات ، من الممكن دائماً أن نصريح عن مواضيع من نوع شكلي ونعطيها إلى متغير وصفي للدالات (شكلية) .

الجدول T2 يعطي قاعدة بالنسبة للتصريح عن أنواع وحدة شاملة ، أي : عند معرفة المؤثر 0 الذي نحتاج إليه في الخوارزم ، يمكننا أن نجعل هذا الخوارزم شاملاً ، وذلك بتحديد النوع الشكلي مع التصريح الأقل دقة الذي يحتوي على 0 .

تتميز « الدقة » في مواصفة شكلية عن النوع بواسطة معيارين يمكن أن يتطابقا . وإذا أشرنا بواسطة S1 و S2 إلى المواصفات الشكلية للأنواع ، نحصل على المعايير :

generic-type-definition	signification	opérations éventuellement utilisées dans l'unité générique et exigée pour le type effectif	
(< >)	نوع عِزاً	خاصيات 3.5.5	عمليات منطقية
range < >	نوع صحيح	attributs 3.5.5	إنشاء جبري
delta < >	نوع حقيقي	attributs 3.5.10	+ , -
digits < >	نوع حقيقي	attributs 3.5.8	*, /, mod, rem, **
array-type-definition	نوع جدول	attributs 3.6.2	ABS
access-type-definition	نوع مرجعي		عمليات : التأشير ، الإلتحام ، تخصيص ، تعيين ، تعادل (مشروط بمعالجة العناصر)
private			تخصيص ، تعيين ، تعادل
private limited			تعيين ، تعادل

T2 العمليات المرتبطة بتصريحات الأنواع

S1 (C1) هو أكثر دقة من S2 <=> S1 يتطلب أكثر الخواص والمؤشرات أكثر من S2 على الأنواع الفعلية .

S1 (C2) أكثر دقة من S2 <=> جميع الأنواع الفعلية التي تناسب S1 ، تناسب أيضاً S2 ، ويوجد أنواع تناسب S2 بدون أن تناسب S1 .

الترتيب الدقيق المناسب للمعيار C1 نحصل عليه من الجدول T2 وهو موجود في الجدول T3 .

نشر هنا إلى قيديين على مواصفة الأنواع :

- النوع شكلي مع يُعَيَّن لا يمكن أن يكون من نوع خاص .

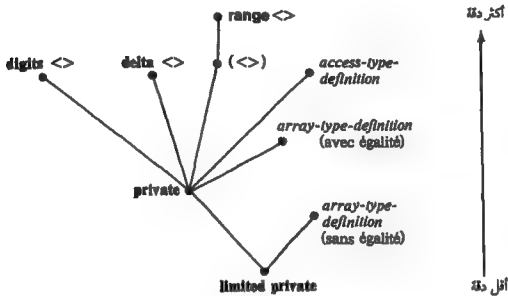
- مواصفة دلائل جدول مُلَزَم لا يمكن أن تتم إلا بواسطة تأثير عن النوع :

generic

type INDEX is (<);

type TAB1 is array (INDEX) of INTEGER ;

type TAB2 is array (1..INDEX'LAST) of INTEGER ;



T3 - الترتيب الدقيق حسب C1

مثال رقم 5 - تطبيق لمعيار الدقة
لفترض جسم الإجراء التالي :

```
procedure EXCHANGE (U, V : in out ELEM) is
  T : ELEM ;
begin
  T := U ; U := V ; V := T ;
end EXCHANGE ;
```

لا نحتاج للنوع ELEM إلا لعملية التعيين ، المواصفة الأقل دقة : limite and private لا تناسب أبداً لأنها لا تحتوي أبداً على « = : » ، وعلى العكس ، من الممكن التصريح عن ELEM private ، حيث :


```
generic
type ELEM is private;
procedure EXCHANGE (U,V : in out ELEM);
```

المراقبة :

```
generic
type ELEM is (<>);
...
```

كانت ستكون فارغة ، ولكن أقل عمومية من ذلك النوع أعلاه لأن ECEM كان سيعتبر كنوع مجزأ ، ولن يتم قبول إلا أنواع كهذه كمتغير فعلي .

مواصفات المتغيرات الشكلية للبرامج الثانوية

مواصفة البرامج الثانوية تدل على اسم أو على رمز المؤثر الشكلي ، أو على أنواع المتغيرات الوسيطة وربما النتيجة [MR 12.13] .

ملاحظة :

قد تندش لوجود معرفات للمتغيرات (في المثل رقم 3) .

with function IMAGE (E: ENUM) return...

المعروف يبدو وكأنه خامض (أنظر مواصفة الإجراءات الشكلية في Algol 68) . وفي الحقيقة ، قد يكون ضرورياً ويسبب قاعلة الرؤية التي تسمح في الجسم بنداء IMAGE بالشكل التالي :

IMAGE (E → SUCC (X))

10.2.4 متغيرات فعلية وقواعد التناسب

قواعد للمواضيع أو القيم

إذا كان متغير الوحدة الشاملة in أو out فإن المتغير الفعلي في التوليد يجب أن يناسب تخصيص المتغير الشكلي كما لمتغيرات البرامج الثانوية . بشكل خاص ، إذا كانت الصيغة in out ، فإن المتغير الفعلي يجب أن يكون متحوّلة .

قاعدة بالنسبة للموضوعات والقيم

من الممكن أن نراجع الجدول T2 وأن نعطي لكل تحديد الأنواع المقبولة فعلياً لكل مواصفة . يُشار إلى التوافق في الجدول T4 .

generic-type-definition	types effectifs compatibles
(<>)	نوع مرقم أو نوع صحيح
range <>	نوع صحيح
delta <>	نوع حقيقي مع إلزام بالفاصلة الثابتة
digits <>	نوع حقيقي مع إلزام بفاصلة متحركة
private	مهما يكن النوع الذي يتم فيه التخصيص والتعامل
limited private	أي من الأنواع (بما فيه النوع مهمة)

T4 - تناسب نوع شكلي - نوع فعلي [MR 12.2, 3.2-5]

يمكن لمواصفة من نوع array أو access ، حيث بعض مركباتها هي شكلية ، أن تظهر في المتغيرات الشاملة ؛ هذه المركبات هي محددة بشكل مسبق . التناسب بين النوع الفعلي والنوع الشكلي هو بسيط : نستبدل في النوع المركب الشكلي الأنواع الشكلية بواسطة قيمتها من نوع فعلي ويجب أن نحصل على النوع المركب الفعلي [MR 5 « 12.3.4] . بشكل خاص ، فالنوع جدول بالزام لا يمكن أن يتناسب مع النوع جدول بدون إلزام ، والعكس بالعكس .

قواعد تناسب البرامج - الثانوية

هذه القاعدة تشبه تلك المعملة في الأنواع المركبة :

- الأنواع الشكلية لمواصفة البرنامج الثانوي هي مستبدلة بواسطة الأنواع الفعلية .
- نحصل على المواصفة التي يجب أن تكون متطابقة مع مواصفة البرنامج الثانوي الفعل -
متغيرات من نفس النوع وب نفس الصيغة ، والنسبة للدوال ، نوع النتيجة الشبيهة - ما
عدا معرفات المتغيرات الشكلية التي يمكن أن تكون مختلفة (أنظر الملاحظة 10.2.3.3) .

يمكن أن يكون المتغير الوسيط الفعلي للبرنامج الثانوي عبارة عن entry .
يمكن إعطاء جدول معاني إمكانية الصيغة التي تتبع المواصفة :

rien	المتغير الوسيط هو إلزامي
is <>	المتغير الوسيط الفعلي هو اختياري والصيغة بالغلط هي عبارة عن برنامج ثانوي ينشئ الاسم كالمواصفة الشكلية ، في مفهوم التوليد .
is "name"	المتغير الفعلي هو اختياري والصيغة بالغلط هي باسم «name» ، في مفهوم التوليد .

10.2.5 إنشاء النماذج

توليد النماذج يمكن أن يتحدد بواسطة قاعدة النسخ التي تقوم بمجموعة تعابير أخرى للغة . هذه القاعدة المنطقية يمكن أن تشرح بعض القيود المحملة إلى تعريف الوحدات الشاملة ، ويجب أن تسمح بفهم لماذا بعض أنواع التوليد هي مسموحة . طريقة التوليد والنسخ موصوفة في ما يلي :

لنفترض التصريح عن الوحدة الشاملة U .

- مفهوم التصريح

تعلي جميع فئات المتغيرات - **generic**
OBJETS_F_NON_INIT : in TYPE1 ;
OBJETS_F_INIT : in TYPE2 := <expr> ;
VARIABLES_F : in out TYPE3 ;
type TYPE_F is <spécification_de_type> ;
with procedure PR_F (<paramètres>) is **OPTION_PR** ;
 < مواصفة الوحدة الشاملة U > ;
 < > جسم الوحدة الشاملة U (إذا كان ضروريا) .

توليد نموذج الوحدة الشاملة U يمتاز بالشكل التالي :

- مفهوم التوليد

<unité> UU is new U
 (OBJETS_F_NON_INIT => <expr1> ,
 VARIABLES_F => VAR,
 TYPE_F => INDICATION_DE_TYPE,
 PR_F => NOM_DE_PROC);

التوليد يعادل الإنشاء التالي :

- تعريف مفهوم تصميم النموذج

OBJETS_F_NON_INIT : constant TYPE1 := <expr1> ;
OBJETS_F_INIT : constant TYPE2 := <expr> ;

-- si OBJETS_F_INIT a une valeur à la génération d'exemple, c'est
 -- dernière qui est prise à la place de la valeur par défaut
 VARIABLES_F : TYPE3 renames VAR ;
 subtype TYPE_F is INDICATION_DE_TYPE ;
 procedure PR_F (<paramètres>) renames NOM_DE_PROC ;

٢٠- تصميم نسخة من U

٢١- نسخ المواصفة U حيث المرفق U مستبدل بواسطة UU

٢٢- نسخ جسم U حيث المرفق U مستبدل بواسطة <UU

- في باقي البرنامج ، فقط التصريح UU هو مرفق .

ملاحظة :

يتم تقويم المتغيرات الوسيطة الشاملة في مفهوم التوليد ، إضافة إلى تعابير الأعداد
 والتهيئة بالخط ، إذا جرى استعمالها وعلى العكس ، جميع المتحولات الأخرى العامة تربط
 بمفهوم التصريح عن الوحدة الشاملة .

10.3 أمثلة

10.3.1 تكامل الدالة

التقرير يعطي [MR 12.4] مثلاً عن رزمة شمولية ، تُعالج كزجلة ، وبعد ذلك
 كنوع مجرد . وهذا يناسب بعض الاستعمال للشمولية . سنعطي هنا مثلاً آخر ، هو مثال
 عن دالات في متغيرات وسيطة . يتعلّق بحساب تكامل بواسطة مربع منحرف (trapeze)
 بفسحات متساوية . الصيغة الرياضية للتكامل هي :

$$\int_a^b f(x)dx = \Delta x [f(a)/2 + f(a + \Delta x) + \dots + f(b)/2]$$

حلّ آدا : البرنامج الثانوي ذو متغير وسيطي هو دالة ، هذا البرنامج الثانوي يجب
 أن يكون شاملاً ؛ المتغيرات الوسيطة الأخرى يمكن أن تكون أو لا تكون ذات طبيعة
 شمولية . في المثال ، هي ليست شمولية ؛ أي إن المثال هو دالة تكامل لدالة معينة ، يأخذ
 حدود التكامل والخطوة كمتغير وسيطي .

generic

with function F (X : FLOAT) return FLOAT ;

function INTEGRATION (A, B : FLOAT ; N : INTEGER) return FLOAT ;

function INTEGRATION (A, B : FLOAT ; N : INTEGER) return FLOAT is

SOMME, DELTAX, X, Y, : FLOAT ;

M : INTEGER ;

begin

SOMME := 0.0 ; M := 1 ;

DELTAX := (B - A) / FLOAT (N) ;

X := A ;

```

loop
  Y := F(X);
  if M = 1 or M = N then Y := Y/2.0 end if;
  SOMME := SOMME + Y;
  X := X + DELTAX; M := M + 1;
  exit when M > N;
end loop;
return SOMME * DELTAX;
end INTEGRATION;

- utilisation :
declare
  function FONTRI (X : FLOAT) return FLOAT is ... end FONTRI;
  Z, W : FLOAT;
  function INTEGRATION_DE_FONTRI is now INTEGRATION (FONTRI);
begin
  ...
  Z := INTEGRATION_DE_FONTRI (0.0, W, 10);
end;

```

10.3.2 النوع المجرد الشامل

هذا المثل يدل على إن التصريح من النوع المجرد الشامل يمكن أن يتطلب متغيرات وسيطة عبارة عن دالات .

- تعريف رزمة شاملة للنوع ENSEMBLE حيث ELEM هو نوع العناصر ، EQ هي دالة التعامل بين هذه العناصر وAFF هي دالة التمييز .

```

generic
  type ELEM is limited private;
  with function EQ (A, B : ELEM) return BOOLEAN;
  with procedure AFF (A : out ELEM, B : ELEM);
  package ENSEMBLES is
    type ENSEMBLE is limited private;
    procedure INSERER (S : in out ENSEMBLE, E : in ELEM);
    procedure ENLEVER (S : in out ENSEMBLE, E : in ELEM);
    function EST_VIDE (S : in ENSEMBLE) return BOOLEAN;
    function APPARTIENT (S : in ENSEMBLE, E : in ELEM) return BOOLEAN;
    function EQ_ENS (S1, S2 : in ENSEMBLE) return BOOLEAN; -- égalité des ensembles
    procedure AFF_ENS (S1 : out ENSEMBLE,
                      S2 : in ENSEMBLE); -- affectation d'ensemble par copie de liste
  private

```

- تمثيل المجموعة هو لائحة بالعناصر للتمية هذه للمجموعة .

```

type ELEM_ENS;
type ENSEMBLE is access ELEM_ENS;
type ELEM_ENS is

```

```

record VAL : ELEM ;
      SUTVANT : ENSEMBLE ;
end record ;
end ENSEMBLES ;

```

package body ENSEMBLES is

- تعريف أجسام الإجراءات والدوال حيث الدالة لشكلية EQ والإجراء AFF هي ضرورية

```

end ENSEMBLES ;
generic                                     - déclaration de l'affectation comme procédure
  type T is private ;
  procedure AFF_SIMPLE (I : out T, J : in T) ;
  procedure AFF_SIMPLE (I : out T, J : in T) is
begin
  I := J ;
end AFF_SIMPLE ;

```

- استعمال الرزمة

```

declare
  procedure AFF_ENTIER is new AFF_SIMPLE (INTEGER) ;
  package ENS_ENTIER is new ENSEMBLES (INTEGER, "=", AFF_ENTIER) ;
  -- paquetage des ensembles d'entiers

  use ENS_ENTIER ;
  package ENS_ENS_ENTIER is new ENSEMBLES (ENSEMBLE, EQ_ENS, AFF_ENS) ;

```

- رزمة من مجموعات من الأعداد الصحيحة حيث التماثل والتعيين هي تلك الخاصة بـ ENS-ENTIER

```

use ENS_ENS_ENTIER ;

```

- الدوال والإجراءات في المثلين هي عملة بشكل زائد فقط
- معرف النوع ليس مرئياً بشكل مباشر

```

S : ENS_ENS_ENTIER.ENSEMBLE ;
E1, E2 : ENS_ENTIER.ENSEMBLE ;

```

- S هو عبارة عن مجموعة من مجموعات الأعداد الصحيحة
- E1 ، E2 هي مجموعات أعداد صحيحة

```

begin
...
if EST_VIDE (S) then ... end if ;
INSERER (E1, S) ;
INSERER (S, E1) ;
...
end ;

```

- استعمال الرزمة ENS-ENTIER
- INSERER — من ENS-ENTIER
- INSERER من ENS-ENS-ENTIER

10.3.3 فرز شامل لأحد الجداول

يتمتع إجراء الفرز بمتغيرات وسيطة شاملة عبارة عن نوع العناصر ، علاقة الترتيب على هذه العناصر ، نوع الجدول ونوع مؤشر الجدول .

```

generic
  type ELEM is private ;
  with function INFEG (A, B : ELEM) return BOOLEAN ;
  type INDICE is (<>) ;
  type TABLEAU is array (INDICE) of ELEM ;
  procedure TRI (T : in out TABLEAU) ;
  procedure TRI (T : in out TABLEAU) is
    - définition du corps de la procédure que l'on ne donne pas ici
  end TRI ;

- création d'exemplaires de la procédure TRI :
type INTERVALLE is range 1..100 ;
type COULEUR is (BLEU, ROUGE, JAUNE, VERT, BLANC) ;
type TAB1 is array (INTERVALLE) of INTEGER ;
  - type d'un tableau contraint d'entiers
type TAB2 is array (COULEUR) of INTEGER ;
procedure TRI_CROISSANT is new TRI (INTEGER, "<=", INTERVALLE, TAB1) ;
procedure TRI_DECREOISSANT is new TRI (INTEGER, ">=", INTERVALLE, TAB1) ;
procedure TRI_CROISSANT is new TRI (INTEGER, "<=", COULEUR, TAB2) ;
  - les exemplaires peuvent posséder le même nom s'il n'y a pas d'ambiguïté
procedure TRI_F is new TRI (INTEGER, "<=", INTERVALLE, TAB2) ;
  - définition illégale d'exemplaire car INTERVALLE n'est pas le type indice de TAB2

```

الامثلة يمكن أن تحتوي على نفس الاسم إذا لم يكن هناك أي إيهام - = < = > Procedure TRI if is new
TRI (INTEGER,

- تعريف غير مسموح للنموذج لأن INTERVALLE ليس هو نوع المؤشر للجدول TAB2

10.4 تقسيم

10.4.1 ملاحظات نحوية أو حول إمكانية القراءة

الحجة المتقدمة للمؤلفين حول لغة آدا هي إمكانية قراءة البرامج . في ما يتعلق بالشمولية ، بعض النقاط تتصادم مع هذه الحجة .

● موقع المتغيرات الفعلية عند توليد المثال هو سيء الاختيار لأنه لا يناسب أبداً موقعها في المواصفة ، ومن جهة أخرى يمكن أن يتطابق بسهولة مع متغيرات النداء بالنسبة للبرامج - الثانوية .
تابع للمثال رقم 1

```

type A is new INTEGER ;
B : A ;
function SQUARE is new SQUARING (A) ;
  SQUARING هي توليد للمثال مع تنوع A والمعملية
  - = * مشتقة من عمليات INTEGER
  - = ! QUARE (B) هو نداء للمدالة

```

begin

B := SQUARE (B) ; -- SQUARE (B) est un appel de fonction
end ;

من الممكن تصور الكتابة كما يلي :

function SQUARE is new (A) SQUARING ;

أو قد تكون :

function SQUARE is new [A] SQUARING ;

● في المتغيرات الشمولية ، **with** التي تسبق **function** أو **procedure** ليست كثيرة الدلالة ؛ فهي لا تظهر إلا لرفع الإبهام النحوي لأن الوحدة الشاملة يمكن أن تكون مواصفة برنامج - ثانوي .

● من الممكن أن نأسف لطرق الإستعمالات **<>** ويشكل خاص عملية التعبير **<> range** هي ذات معنى لجدول بدون إلزام في حالة التصريح عن نوع جدول ، بينما هي تعني « نوع شكلي صحيح » في المواصفات الشمولية .

generic
type T is range <> ;
type U is array (T range<>) of INTEGER ;
package P is ...

مكلا مواصفة ليست سهلة لا للشرح ولا للفهم

10.4.2 حول مواصفات الأنواع الشكلية

من مواصفات الأنواع الشكلية ، يمكن أن نأسف لغياب المواصفة من نوع فقرة (article) . يمكن أن يمر هذا الغياب لحساب بعض الصعوبة في البرامج . ولو كنا قد أدخلنا هذه المواصفة ، لكانت عمليات إنتقاء الحقول قد إستطاعت أن تتصرف كمعرفات لمتغيرات وسيطية في البرامج الثانوية الشكلية .
مثال رقم 6 : عرض مواصفة من نوع شكلي فقرة .

generic
type T1 is private ;
type T2 is
record

- مواصفة غير مسموحة في آدا


```

. A : INTEGER ;
  B : T1 ;
end record ;

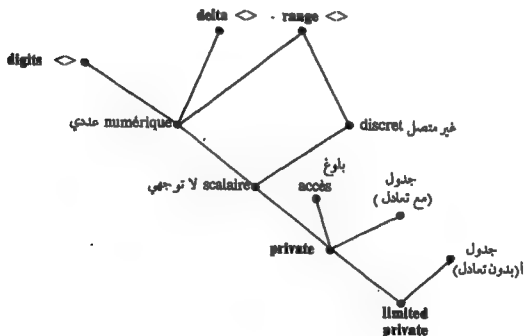
package P is
...
end P ;

-- création d'un exemplaire
type R is
  record
    UN : INTEGER ;
    DEUX : COULEUR ;
  end record ;
package Q is new P (T1 => COULEUR, T2 => R);

```

- رزمة شاملة حيث عمليات إنتهاء
- جدول T2 هي شكلية
- تصريح مقبول :
R يناسب T2

● ويسبب وجود تعبير للمواصفة من نوع مجزأ « < > » ، والذي يُناسب فئة معينة : نوع مُرقّم وصحيح (T1) ، فقد ندهش لعلم حصولنا على المواصفة للطبقات الرقمية والطبقات الساكنة (أنواع مع ترتيب) . وهذا كان سيعطينا الشجرة من الرسم T5 (قارنّه مع T3) .



T5 - ترتيب منطقي لفئات الأنواع

مثال رقم 7

فلنجد الأكبر بين عنصرين ، الجسم يظهر وكأنه :

```
function MAX (U, V : in ELEM) return ELEM is
  R : ELEM ;
begin
  if U < V then R := V ; else R := U ; end if ;
  return R ;
end MAX ;
```

- العمليات المستعملة على ELEM هي :

- التمين

- < > : من نوع ELEM × ELEM → BOOLEAN

ولو كان بتصرفنا أنواع لا اتجاهية ، لكان بإمكاننا كتابة المواصفة الشاملة :

```
generic
  type ELEM is scalar ;      - غير مسموح في أدا
function MAX (U,V : in ELEM) return ELEM ;
```

وبما أن هذا هو غير ممكن ، فإمكاننا إما تعريف ثلاثة نماذج شاملة مع ELEM وعلى التوالي < > digit ، < > delta و < > ، مما يغطي كامل حقول الأعداد اللاتجاهية ، وإما استعمال دالة شكلية في المواصفة لـ < > :

```
generic
  type ELEM is private ;      - بسبب التمين
  with function "<" (X,Y : ELEM) return BOOLEAN is <> ;
function MAX (U,V : ELEM) return ELEM ;
```

أثناء عملية إنشاء للمثال ، وإذا كان النوع ELEM هو لا اتجاهي ، فليس من الضروري إعطاء الدالة لأن المؤثر المحدد < > سيكون مستعملاً :

```
function IMAX is new MAX(INTEGER) ;      - صيغة بالغلط
function IMIN is new MAX(INTEGER, ">") ;  - نموذج آخر
```

10.4.3 ملاحظة حول التوليد الصريح للنماذج

لغة Ada ، وعلى عكس بعض المفاهيم الأخرى للشمولية ، أو لتمدد الأشكال (انظر 10.1.2) ، إختارت أن تجعل عملية توليد النماذج لوحدة شاملة واضحة وجلية (التعليمة new) . وهذا هو ضروري بالنسبة للرمز ، ولكن البرامج الثانوية يمكن أن تكون مولدة أوتوماتيكياً في لحظة النداء لتوليد ضمني . [MI: 1.3.4.1] تصرّطويلاً ، على

هذا الأمر بحاجة أن إنشاء النماذج هو قسم من تصميم التصريحات وليس من تنفيذ التعليمات وإن التوليد الضمني يؤدي إلى صعوبات في التعريف بسبب التحميل الزائد الممكن . فلنفحص الحالات المحدودة حيث هذا التوليد الممكن يخلق المشاكل .

مثال رقم 8

- حالة حيث التوليد هو ضمني : عندما يكون البرنامج التائي متكرراً ، فالنموذج المختار للنداء الداخلي هو نفسه كالنداء الأول .

```
generic
  with function "*" (A, B : NATURAL) return NATURAL ;
function GENERAL_OP (N : NATURAL) return NATURAL ;

function GENERAL_OP (N : NATURAL) return NATURAL is
  R : NATURAL ;
begin
  if N = 0 then R := 1 ;
  else R := N*GENERAL_OP (N - 1) ;
  end if ;
  return R ;
end GENERAL_OP ;
```

فلنشير إلى أن التكرارية التصالبيه ليست ممكنة إذا لم تكن داخلية :

مثال رقم 9

```
generic
  with function "*" (..) ... ;
function G_OP1 (..) ... ;
-- première fonction générique

generic
  with function "+" (..) ... ;
function G_OP2 (..) ... ;
-- deuxième fonction générique

function G_OP1 (..) ... is
  R : NATURAL ;
begin
  ...
  else R := N*G_OP2 (N - 1) ;
  and if ;
  return R ;
end G_OP1 ;
-- appel de G_OP2 : erreur
```

هناك طريقتان لحل هذه المسألة . الأولى تقوم على تجميع الكل في رزمة شاملة .

```

generic
. with function "+" (..) ...;
  with function "*" (..) ...;
package FONCTIONS is
  function G_OP1 (..) ...;
  function G_OP2 (..) ...;
end FONCTIONS;

```

—solution de l'exemple 9

```

package body FONCTIONS is
  function G_OP1 (..) ... is
    R : NATURAL;
  begin
    ...
    else R := N * G_OP2 (N - 1);
    end if;
    return R;
  end G_OP1;
  function G_OP2 (..) ... is
    ...
  end G_OP2;
end FONCTIONS;

```

هذا قد يكون مضجراً إذا أردنا كتابة دالة شاملة تتعلق بنوعين شاملين محددتين بشكل منفصل كلٍ منهما عن الآخر .
يجب علينا إذا إنشاء نماذج في داخل الوحدة generic : وهذا هو الحل الثاني :

مثال رقم 10

```

generic
  type T is private;
package VECTEURS is -- paquetage des VECTEURS sur T
  type VECTEUR is array (NATURAL range <>) of T;
  ...
end VECTEURS;

```

```

generic
  type T is private;
package MATRICES is -- paquetage des MATRICES sur T
  type MATRICE is array (NATURAL range <>, NATURAL range <>) of T;
  ...
end MATRICES;

```

```

generic
  type T1 is private;
  with function "+" (A, B : T1) return T1;
  with function "*" (A, B : T1) return T1;
  -- paquetage des opérations produits de matrices et vecteurs

```

```

package PRODUITS is
package T1_VECTEUR is new VECTEURS (T1);
use T1_VECTEUR;
package T1_MATRICE is new MATRICES (T1);
use T1_MATRICE;
function "+" (M, N : MATRICE) return MATRICE;
function "*" (M : MATRICE, V : VECTEUR) return VECTEUR;
function "*" (V1, V2 : VECTEUR) return T1;
end PRODUITS;

```

10.4.4 ملاحظات حول بعض النقاط [MRA]

كان الفصل الذي يعالج الشمولية قد تطور كثيراً بين الصيغة 79 [GREEN] والصيغة 80 [MR] . ولقد حملت [MRA] تغييرات لم نأخذها في الحسبان في هذا الفصل . التعديلات الأساسية هي عبارة عن تدقيقات على مدى المرفقات ، وعلى أوالية تصميم التوليد التي جرى تحديثها بالكامل (10.2.5) . فلننجز الملاحظات المستوحاة من هذه الصيغة الجديدة للمساعد المرجعي [MRA] .

● جرى معالجة التصريحات عن المتغيرات الشاملة بشكل عملي في اللغة : مفهوم النوع - الثانوي ، إعادة التسمية . لهذا نخفّض عدد المعاني المجردة أو التصورات ، وهذه نقطة إيجابية . ولكن ، عملية تعريف متغير in بالتصريح عنه ككثافة مهينة بواسطة المتغير الفعال بواسطة التخصيص يلغي إمكانية العبور بواسطة in للقيم من النوع المحدد ، لأن هذه الأخيرة لا تحتوي على التعيين نعود إلى هذا القول كون الصيغ in out in للبرامج الثانوية تعرف بشكل مختلف عن الصيغ in out in للمتغيرات الشاملة .

● التعابير بالغلط ليست مصممة إلا إذا كان المتغير الوسيط الفعلي غائباً في لحظة التوليد . وهذه الدقة مهمة . وعلى العكس ، لا نعرف جيداً فيما إذا كانت المرفقات العامة في وحدة شاملة هي معرفة في إطار التصريح (الصيغة [MR]) ، لأن الجملة [MRA 12.3] تدل على أن المرفق هو علم « ويعني نفس الوحدة في الصيغة وفي النموذج » . إذا كانت قواعد الرواية تعمل بعد « النسخ » معرف لا يعني أبداً نفس الوحدة ، فلا نعرف أبداً إذا كان التصريف يختار الوحدة المرتبطة بالتصريح ، أو إذا كان هناك خطأ . في المثال رقم 2 ، اعتمدنا الحل الأول المتوافق مع [MR] .

● مدى المرفقات هو هكذا كي يحمّد حقن التطبيق للشمولية ، كما يدل المثال رقم

مثال رقم 11

- FONCT هي عملية تبني F كمتنبر وسيطي

```
generic
with function F(N : INTEGER) return INTEGER ;
function FONCT (M : INTEGER) return INTEGER ;
function FONCT (M : INTEGER) return INTEGER is
begin
  if M = 0 then return 1 ;
  else return M * F (M - 1) ;
  end if ;
end ;

-- PLUS_UN sera utilisée dans les exemplaires de FONCT
function PLUS_UN (N : INTEGER) return INTEGER is
begin return N + 1 ;
end PLUS_UN ;
```

-- création d'exemplaires de FONCT
 function CARRE is new FONCT (PLUS_UN) ;
 function FACT is new FONCT (FACT) ; --
 فرمسموح به لأن FACT لا يمكن أن --
 يستعمل طالما أن التصريح عنه ليس مصمماً بشكل كامل .

10.5 خاتمة

جرى إدخال الشمولية في آدا للإجابة على دفتر الشروط لـ DOD ومن الواضح إن هذا التعبير هو مستقل عن التعابير الأخرى من اللغة . في أغلب الحالات ، يجب أن تكون الصيغ كافية .

وبشكل خاص ، فإن الجهد المبذول لتعريف فئات الأنواع مع عملياتها هو كبير . نلاحظ إنه وعلى عكس مفهوم الماكرو-مولد ، فمن الممكن للمستعمل أن يؤمن تصحيح وحداته الشاملة ، والمصرف قادر على أن يقوم في هذا المستوى بأغلب التدقيقات الساكنة . وهذا ما يجب أن يساعد المصرف على عدم فقدان فعاليتها عند معالجته للعمليات الشاملة النوعية .

من جهة المستعمل يُقال إن كتابة الوحدات النوعية الشاملة يتطلب بعض العناية بينها إستعمالها هو بسيط . وهذا الهدف هو مبلوغ في لغة آدا .

الفصل الحادي عشر

التصريف المتفصل

11.1 مدخل

هذا الفصل يُعالج السهولة المقدمة بواسطة آدا في كل ما يتعلق بتصريف البرامج ، والمشاكل التي قد تحدث في مستوى الاستعمال أو في مستوى إنشاء المصمم [MR 10] . تقسيم البرنامج ومعالجته على مراحل بواسطة المصنف هي عملية شائعة . وفي أغلب الأحيان ، لجهة تركيبة المترجم ، ليس بشكل عام القيام بعمليات تدقيق للتماسك بين مختلف وحدات التصريف .

وهذه هي التدقيقات التي نفترض إنشاء مصنف مفصل للغة آدا . هذه الفكرة ، البسيطة ظاهرياً ، تؤدي إلى عدة تعديلات :

- في عادات المستعملين ، الذين يجب عليهم المحافظة على بعض الترتيب في وضع وحداتهم للمترجم .
- في تركيبة المترجمين ، الذين يجب عليهم أن يحتفظوا ببعض المعلومات كي يتم إجراء بعض عمليات التدقيق المذكورة .

11.2 تقسيم البرنامج بغية التصريف المتفصل

يمكن أن يُقسم البرنامج آدا إلى عدة وحدات تصريف . تحتوي وحدة التصريف على :

- مواصفة الرزمة (من المحتمل أن تكون نوعية) .
 - جسم الرزمة .
 - مواصفة البرنامج الثانوي (من المحتمل أن تكون جلية) .
 - جسم البرنامج الثانوي .
 - وحدة - ثانوية .
- وحدات الفئات الأربع الأولى هي وحدات من مكتبة البرامج . الوحدة - الثانوية

هي وحدة منطقية داخلية في وحدة تصريف أخرى (unité m re - وحدة - أم) ، والتي تنفصل عنها بواسطة التصريف . التصريح المناسب لهذه الوحدة - الثانوية يجب أن يكون في القسم الوصفي الخارجي للوحدة الأم . الوحدة - الثانوية يمكن أن تكون :

- جسم الرزمة .

- جسم البرنامج الثانوي .

- جسم المهمة .

وفي النهاية ، تختلف الوحدات (من نفس البرنامج) والخاضعة للمصرف يمكن أن تكون :

- وحدات « مستقلة » .

- وحدات منفصلة ، متداخلة بشكل طبيعي في وحدات أخرى ومقطوعة عن هذه الأخيرة .

هاتان الطريقتان تناسبان طرق التحليل والتطبيقات .

11.2.1 وحدات منفصلة

مثال رقم 1

لنفترض إجرائين متداخلين :

```
procedure P is
  procedure R is
  begin
    ...
  end R ;
begin
  R ;
end P ;
```

المثل أعلاه يمكن أن يكون خاضعاً للمصرف على الشكل التالي :

مثال رقم 2

```
procedure P is
  procedure R is separate ;
begin
  R ;
end P ;
```

- لزومة R


```

separate (P) ;
procedure R is
  begin
    ...
  end R ;

```

الإجراء R هو منفصل عن الإجراء P ولا يمكن أن يُصوَّف إلا بعد P . في كل ما يتعلق بمدى R ، تسير الأشياء كما لو أن جسم هذه الرزمة كان موجوداً في الموقع المشار إليه وكأنه منفصل (أرومة - Souche) . في غياب كل with ، فإن إمكانية الرؤية لدى R تصبح ذاتها إمكانية رؤية أرومتها .

11.2.2 وحدات مستقلة

بخصوص التطبيق ، من الممكن تطوير عدد من الوحدات التي تستعمل لاحقاً في التطبيق .
مثال رقم 3

```

package P is
  ...
end P ;

```

مواصفة P

البلوغ إلى المواصفة P لحصل عليه بواسطة with

```

with P ;
procedure R is
  ...
end R ;

```

ـ هنا نرى كل المواصفة P

الوحدات P و R هي وحدات مستقلة ؛ وليست متداخلة مباشرة ولا غير مباشرة في أية وحدة أخرى . الوحدات المستقلة تدهى وحدات من مكتبة .

11.2.3 النص الكامل لتعريف الوحدة

الجملة with الموضوع في رأس الوحدة المطلوب تعريفها تدل على وحدات المواصفة من المكتبة والصالحة لأن تستعمل بواسطة هذه الوحدة . أسماها مركبات المواصفة الداخلة هكذا ليست قابلة للاستعمال في الوحدة المطلوب تعريفها إلا بالشكل المنقط . ولجعلها مبلوغة مباشرة من الضروري إستعمال use . مشاكل الرؤية الناتجة جرى معالجتها في 7.2.2 .

التراتبية الموجودة بين وحدة مواصفة (مواصفة رزمة أو مواصفة برنامج ثانوي) ،

بين إنشائها (وحدة - جسم) وبين الوحدات الثانوية من هذه الأخيرة تعيدنا إلى القاعدة التالية :

النص الكامل لتصريف وحدة مواصفة ينطبق أيضاً على الوحدة - الجسم المتأنيبة والتي يمكن أن تغنيه بواسطة *with* إضافية . إضافة لذلك ، فإن نص الوحدة الثانوية مبني من خلال النص الكامل للوحدة - الأم ، الأوامر *with* الموضوعية في رأس نص الوحدة - الثانوية .

مثال رقم 4

نفترض وجود وحدات المواصفة X, Y, Z في المكتبة .

```
with X ;
package A is
  ...
end A ;
```

```
-----

with Y ;
package body A is
  ...
  procedure P is separate ; -- souche de la procédure P
  ...
end A ;
```

```
-----

with Z ;
separate (A) ;
procedure P is
begin
  ...
end P ;
```

في هذا المثل :

- النص الكامل لتصريف وحدة المواصفة A يتشكل من وحدة المواصفة X .

- النص الكامل لتصريف الوحدة - الثانوية A يتشكل من وحدات المواصفة A, X, Y .

- النص الكامل لتصريف الوحدة الثانوية P المفصولة عن الوحدة - الجسم تشكل من وحدات المواصفة A, X, Y, Z ، ومن متحولات مركزية للوحدة - الجسم المصرح عنها قبل الأرومة (souche) .

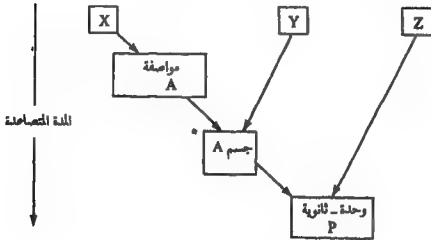
11.2.4 نظام ترتيب الوحدات

بشكل عام ، يجب أن تصريف كل وحدة - جسم بعد وحدة المواصفة المناسبة .
كذلك فكل وحدة - ثانوية يجب أن تصريف بعد الوحدة التي انفصلت عنها .
إستعمال with يفرض أيضاً ترتيباً (جزئياً) للتصريف لأن كل وحدة حاملة لوحدة
من المكتبة (وحدة مواصفة) يجب أن تصريف بعد الوحدة المحمولة .

في المثال رقم 4

- وحدة المواصفة A يجب أن تصريف بعد وحدة المواصفة X .
- الوحدة - الجسم يجب أن تصريف بعد وحدات المواصفة X ، Y و A .
- الوحدة الثانوية P يجب أن تصريف بعد وحدة المواصفة Z وبعد الوحدة - جسم A .

هذه القواعد تحدد ترتيباً جزئياً (لا يفرض أي ترتيب في كل ما يتعلق بالوحدات X, Y, Z) ، نستطيع تخطيطه مع المخطط رقم 1 . كل نظام ترتيب للتصريف يحافظ على هذا الترتيب الجزئي هو صالح . هذا الترتيب الجزئي يسمح أيضاً بتحديد عمليات إعادة التصريف المتتالية والمحتملة عند إعادة تصريف الوحدة .



مخطط رقم 1

11.2.5 نظام ترتيب تصميم الوحدات

قبل تنفيذ البرنامج الرئيسي ، يجري تصميم جميع وحدات المكتبة المستعملة .
وحدات المكتبة هذه هي نفسها المذكورة في الأمر with في البرنامج الرئيسي وفي الوحدات

الثانوية ، هذا ما يحدث أيضاً في الأوامر with من وحدات المكتبة نفسها ، وهكذا دواليك ، بشكل إنتقالي . عمليات التصميم هذه تتم بترتيب يحافظ على الترتيب الجزيئي المحلّد في الأوامر with (الفقرة السابقة) .

ولكن ، لو افترضنا إن طلب البرنامج الثانوي المحلّد في الرزمة X يتم خلال تصميم جسم رزمة أخرى (أي خلال تصميم القسم الوصفي أو خلال تنفيذ قسم الأعداد في هذه الرزمة) ، يجب أن يتم تصميم جسم الرزمة X أولاً .

البرنامج هو غير صالح إذا لم يوجد ترتيب متماسك وهو مغلوط إذاً ، وعندما يكون هناك عدة طرق ترتيب ممكنة ، فإن معناه يتعلّق بالترتيب النهائي المختار .

11.3 تقييم التعريف المنفصل

11.3.1 تقسيم البرنامج الثانوي

التقسيم المنفصل للغة أدا يحلّل تحسينات غير قابلة للمناقشة بالنسبة للموجود . سنقوم بمحاولة إيجازها من خلال الأمثلة التالية .

مثال رقم 5

I	II	III
<pre> procedure P is procedure R is ... end R ; begin R ; end P ; </pre>	<pre> procedure P is procedure R is separate ; begin R ; end P ; <hr/> separate (P) procedure R is begin ... end R ; </pre>	<pre> procedure R is begin ... end R ; <hr/> with R ; procedure P is begin R ; end P ; </pre>

الأمثلة I و II هي متساوية . إذاً في هاتين المثلين ، لا يستعمل الإجراء R شيئاً مما هو مصرّح عنه في P ، هما إذاً متساويان مع المثل III . على هذه الأمثلة الثلاثة يمكن للمصرّف أن يقوم بنفس التديقات . فليمر ما يجري عندما نضطر لتعديل هذه الأمثلة :

	(I)	(II)	(III)
تعديل في جسم R	يجب أن يُعاد تصريف الكل	فقط R يجب أن يُعاد تصريفه	يجب أن يُعاد تصريف الكل
تعديل في جسم P	يجب أن يُعاد تصريف الكل	يجب أن يُعاد تصريف الكل	فقط P يجب أن يُعاد تصريفه

11.3.2 إنشاء نصوص التصريف

تتمتاز إمكانية وضع with قبل وحدة - ثانوية (حالة الإجراء P في المثال السابق) بإيجابية عبارة عن إمكانية - الوحدة الثانوية من بلوغ أكثر من وحدة في المكتبة من التي يمكن أن تبلغها الوحدة - الأم . إدخال نص وحدة كهذه في الوحدة - الأم وفي نقطة الفصل لا يمكن أن تكون بدون أخطار .

مثال رقم 6

فلنفترض رزمتين من الرييلة A و B هله المواصفات التالية :

```
package A ;
...
package B ;
  X : INTEGER ;
end B ;
end A ;
```

```
package B ;
...
X : INTEGER ;
...
end B ;
```

فلنكتب الآن الرزمة التالية :

```
with A ;
package body C ;
  use A ;
  ...
  procedure Q is separate ;
  ... B.X ...      - A.B.X
end C ;
```

```
with B ;
separate (C) ;
procedure Q is
...B.X...      -- B.X
end Q ;
```

في الإجراء Q ، تعني B.X المتحولة X من وحدة المكتبة B . في الرزمة C ، B.X تعني المتحولة X من الرزمة B الداخلة في الرييلة A . إعادة إدخال Q في مكان وموقع أرومتها (رأسها) يفرض وضع with في رأس الرزمة C . الأمر use A يصبح غير عاما

بالنسبة للرزمة B الداخلة في A لأنه يوجد الآن وحدة مرئية مباشرة باسم B . هكذا فإن B.X لا يمكن أن تُنشأ A.B.X ، ولكنها تعني المتحولة X في وحدة المكتبة B .

فلنشير الآن إلى أنه ، ولتسهيل عمليات إعادة الإدخال هذه ، فإن إمكانية رؤية وحدة مفصولة هي شبيهة بإمكانية رؤية أرومتها (في غياب with أمام جسم هذه الوحدة) . هذا ما يجبر المترجم على المحافظة على النص الكامل المناسب لكل أرومة لأن وحدتين ثانويتين من نفس الوحدة يمكن أن يحصلان ، حتى في غياب with ، على نصوص تصريف مختلفة .

مثال رقم 7

```
procedure T is
...
procedure P ( . . ) is separate ;
procedure Q ( . . ) is
...
begin
...
end Q ;
procedure R ( . . ) is separate ;
...
begin
...
end T ;
```

الإجراء T.Q ينتمي إلى نص التصريف T.R ولكن ليس إلى نص T.P . إذاً هذا حقاً مفيد ؟ حلّ الفصل المنتظم بين المواصفة والجسم (أنظر 6.1.4) كان ينمّج ، في حالة التصريف المنفصل ، بإجراء إقتصاد في الأرومة وتأمين نفس النص لجميع الوحدات - الثانوية .

11.3.3 ترتيب تصميم الوحدات

في كل ما يتعلق بترتيب تصميم الوحدات ، فإن [MR] هو أكثر وضوحاً . وهو يقتصر على قاعدتين وتعريفين سنذكرهما هنا وهما [MR 10.5] .

القاعدة 1 : « تصميم الوحدات يجب أن يحافظ على الترتيب الجزئي المفروض بواسطة with » .

القاعدة 2 : « تصميم أجسام الرزم يجب أيضاً أن يحافظ على علاقات التبعية الناتجة عن الأفعال المتوقعة خلال تصميم هذه الأجسام » .

تعريف 1 : « البرنامج هو illegal (غير مشروع) إذا لم يكن في الإمكان إيجاد أي نظام ترتيب للتصميم (في حالة وجود دوران في العلاقات التبعية) .

تعريف 2 : « البرنامج هو مغلوطة *errone* في حالة وجود عدة ترتيبات ممكنة للتصميم وإن البرنامج يتعلق بالترتيب المختار » .

من الواضح أنه لا القواعد ولا التعريفات ، لا تسمح بالإجابة على السؤال الأساسي : في لحظة معينة نحلّد نظام ترتيب التصميم ؟

الأجوبة الممكنة على هذا السؤال هي ثلاثة :

- نظام ترتيب التصميم يثبت عند التصريف .
- نظام ترتيب التصميم يثبت عند التنفيذ .
- نظام ترتيب التصميم يثبت بعد التصريف وقبل التنفيذ .

نظام ترتيب التصميم المثبت عند التصريف

نظام ترتيب التصميم المختار هو نظام ترتيب تصريف الوحدات . هذا الحلّ الفطري (لا يحافظ على القاعدة 2 ولكن سنرى إن هذه القاعدة هي غامضة) له فائدة كبيرة : البساطة .

نظام ترتيب التصميم المثبت عند التنفيذ

هذا الحلّ يغطي الحالة الأكثر عمومية . يتعلّق ذلك بالبداية بتنفيذ البرنامج وتصميم الوحدات حسب الحاجات . عندما يكون البرنامج *illegal* ، (غير مشروع) ، فلهذا الحلّ فائدة تكمن في عدم استخدام سوى الوحدات المستعملة فعلياً خلال تنفيذ البرنامج . ولكن ، ومع إن هذا المفهوم هو كافٍ من الناحية النظرية ، وهو شديد الفعالية فإن له عدة سيئات :

- النظام سيحمل عند التنفيذ أعباء التصميم « كشف » الترتيب الجزئي هو عبارة عن منهاج أو برنامج غير مبدل .

- أقسام إعداد وتهيئة رزم التطبيق يمكن أن تحتوي على عدد كبير من التعليمات (ليس لها علاقة مع الأعداد) . ومن المزعج دائماً معرفة عدم وجود ترتيب يتعلّق بالتصميم (أي إن البرنامج هو *illegal*) بينما تكون المدة الكبيرة للتصريف قد انتهت .

نظام ترتيب التصميم المثبت بعد عمليات التصريف وقبل التنفيذ :

هذا الحلّ ، الذي يبدو وكأنه عملية تنقيح كلاسيكية للأربطة ، هو الذي يكشف من عدة ملاحظات مختلفة [GI10.5] ، وهو نفسه الذي يبدو وكأن مصممي اللغة يتوجهون نحوه . لهذا الحلّ إذاً ، سنقوم بالتحليل الدقيق للقواعد والتعريفات المذكورة سابقاً :

برنامج illegal (غير مشروع)

فلنفترض المثال السابق :

مثال رقم 8

```
package A is
...
  procedure P ;
...
end A ;
-----
package B is
...
  procedure F ;
...
end B ;
-----
with A ;
package body B is
...
  procedure F is ... end F ;
...
begin
  A.P ;
end B ;
-----
with B ;
package body A is
...
  procedure P is ... end P ;
...
begin
  B.F ;
end A ;
```

يوجد ، هذه الوحدات الأربع ، ترتيب مختلف للتصريف . وعلى العكس ، فلا يوجد أي ترتيب ممكن للتصميم لأن كل نداه A.P أو B.F في القسم تعليمات جسم الرزمة B أو A يتطلب أن يكون الجسم الآخر للرزمة قد جرى تصميمه حسب الإمكان

(حسب القاعدة 2) . البرنامج

هو إذا غير مسموح . فلننظر إلى

ما يجري عندما نقوم بتعديل

الوحدة الأخيرة من مثالنا

(جسم A) .

```
with B ;
package body A is
  X : constant BOOLEAN := FALSE ;
  procedure P is ... end P ;
...
begin
  if X then B.F end if ;
end A ;
```


في هذه الحالة ، يبقى البرنامج « غير مسموح » لعدد كبير من المصروفات . ولكن قواعد المائلة [MR 10.6] تسمح للمصروفات بعدم توليد كود لتعليمات من النوع «if ...FALSE then» . بعض المصروفات ، وعند حسابة X ، تقوم على إلغاء التعليمة «if X then B.F. end if» . البرنامج لن يكون illegal لأن الدوران سيخفي . مفهوم البرنامج «illegal» هو إذا مربوط بالمصرف مما يضر بإمكانية النقل : أي إن البرنامج العامل على مكنة سيكون غير عامل على مكنة أخرى .

برنامج مغلوطة

من غير الممكن إكتشاف برنامج مغلوطة بشكل ساكن . إذا كان هناك ، برنامج ، وحدة ترتيبات مكنة للتصميم ، فالاختيار يجب أن يتم بشكل عشوائي (إذا قمنا بإعادة تصريف وحدة بدون تعديلها ، بينها ترتيب التصميم يمكن أن يُعدّل) أو بشكل كيفي (إذا قمنا بتصريف وحدة دون تعديلها ، فإن نظام ترتيب التصميم لا يتعدّل ، ولكن مصرّفون مختلفون يمكن أن يقوموا باختيار مختلف) . لذلك نجد هنا مشكلة حول إمكانية النقل .

ولكن هناك مشكلة خطيرة : فليس مستعمل لغة آدا أية وسيلة لترتيب وحدتين - جسم - لا يتمتعان بأية علاقة تبعية جلية .

```
package body A is
  X : INTEGER ;
  ...
begin
  GET (X) ;
end A ;
```

```
package body B is
  Y : CHARACTER ;
  ...
begin
  GET (Y) ;
end B ;
```

ليس للبرزمتين A و B أية علاقة تبعية بينهما . وحسب ترتيب التصميم المختار (A وبعد ذلك B أو B وبعدها A) فتعليمة القراءة الأولى ستحاول قراءة سمة أو عند صحيح . للبرنامج حظ على إثنين بالانقطاع بتبعية غلط في القراءة . .

نظام ترتيب التصميم

القاعدة 2 تتعلق فقط بجسم الرزم . لا يُقال شيئاً عن العلاقات التي قد توجد بين وحدات التصريف الأخرى مما قد يؤدي إلى علاقات حساسة كما يبرهن المثال التالي المأخوذ من [GI 10.5] .

```

package A is
  ...
  X : INTEGER ;
end A ;
-----
with A ;
package B is
  ...
  Y : INTEGER := X ;
end B ;
-----
package body A is
  ...
begin
  X := 0 ;
end A ;

```

في هذا المثال ، لا يوجد أية علاقة تبعية بين مواصفة الرزمة B وجسم الرزمة A .
أنظمة ترتيب التصميم الممكنة لهذه الوحدات هي :

(1)	(2)
مواصفة A	مواصفة A
B مواصفة	A جسم
A جسم	B مواصفة

في الحالة (1) ، تتمتع B.Y بقيمة غير عتدة بينا في الحالة (2) فهي ستكون بالقيمة 0 . هذا البرنامج هو إذاً مغلو ط .

11.4 خاتمة

إذا كنا نرضى فيما يخص بمبادئ التصريف المنفصل وقواعد ترتيب المصريفات المختلفة لوحدة البرنامج ، فالأمر ليس كذلك بالنسبة لتحديد نظام ترتيب تصميم هذه الوحدات .

في الحقيقة فإن مسألة نظام ترتيب التصميم هو مشكلة أساسية في الوقت الحالي . وهناك مفهومان تقريران لذلك :

- مفهوم ساكن . يُحدّد نظام التصميم عند التصريف ، تُحمّل جميع الوحدات (حتى تلك التي تكون غير مطلوبة) وتُصمّم حسب نظام الترتيب هذا قبل أن يبدأ التنفيذ الحقيقي

للبرنامج . هذه التقنية تشبه التنقيح الكلاسيكي .

- مفهوم ديناميكي : تحمل كل وحدة وتصرف عند الحاجة لها (أي بشكل متأخر ، مما يتناسب مع مفهوم delay binding time أو الربط الديناميكي) ، ويُستخدَم لذلك وحدة إنطلاق مميزة .

ولكن مهما يكن الاختيار ، فمن الواضح إن بعض المشاكل تبقى في قسمها الأكبر بسبب الصعوبات التي نلتقيها في استعمال أقسام إعداد وتهيئة الرزم . هذه الأخيرة هي غير قابلة للاستعمال بطمأنينة إلا لمعالجة المعطيات الخاصة بالرزم دون العودة إلى الوحدات الخارجية . من المؤسف أن تكون اللغة غير معتمدة بالنسبة للقيود التي تسمح بتأمين هذا الأمان .

إن صعوبة تحديد نظام ترتيب التصميم بالنسبة لبرنامج آدا تبرّر التعديلات الجارية في المساعد - المرجعي عند مراجعة المعيار ANSI [MRA] .

القاعدة (2) (المناسبة لتصميم جسم الرزمة) جرى إلغاؤها . ويشار بدلاً عنها إلى ان الوحدات - الجسم يمكن أن تصمّم في الأخير (أي بعد جميع وحدات - المواصفة) وفي أي ترتيب ممكن (لا نحسب حساباً للعلاقات التبعية بين وحدات - الجسم) . وبكلمة أخرى ، فقط صعوبات التصميم بالنسبة لوحدات البرنامج هي تلك الناتجة عن التصريف المنفصل .

كل ترتيب يحافظ على الترتيب الجزئي المفروض من with يصبح ترتيباً صالحاً للتصميم (وبشكل خاص فإن ترتيب عمليات التصريف هو ترتيب تصميم صالح) . مشكلة العلاقات التبعية بين وحدات الجسم جرت إعادة إثارتها في مستوى المستعمل بواسطة إدخال الكلمة ELABORATE التي يمكن بواسطتها فرض ترتيب للتصميم بين مختلف أجسام الرزم .

الفصل الثاني عشر

وسائل التكييف

في هذا الفصل ، سندرس مختلف الوسائل في لغة آدا التي تسمح للمبرمجين بتكييف برنامج عام مع شروط محيط محدّد للتنفيذ . سنعيد أولاً ذكر مختلف المهام التي قد تؤثر على عمل تكييفي ، إضافة إلى الإمكانيات للعمل بها في نظام برجة معين : نستطيع إذا تركيز وتقييم المبادئ العامة في لغة آدا . سنحلّل بعد ذلك تفصيل مختلف الإمكانيات المقدّمة بواسطة هذه اللغة لعمليات التكييف هذه .

الأقسام المعالجة

مساعدة مرجعي [MR, MRA] :

الخواص : 3.3 ، 3.5 ، 3.58 ، 3.51 ، 3.62 ، 3.72 ، 4.14 ، 13.7 الملاحق

. F و A

الدرائع : 2.8 ، 4.8 ، 6.3 ، 9.8 ، 11.7 ، 13.1 ، 13.7 ، 13.9 ملاحق B ،

. F

مواصفات التمثيل 13.6 إلى 13.1

مميزات تتعلق بالمحيط : 13.51 ، 13.7 إلى 13.10 ، ملاحق C, F .

مساعدة في الشرح [ME] : الفصل 14 بالكامل .

12.1 مدخل

12.1.1 فائدة وسائل التكييف

تعتبر لغة البرمجة ذات مستوى عالٍ إذا كانت تسمح بحلول مُجرّدة للمسائل المعلوماتية . كما يعرف كل منا ، فإن الحلّ المُجرّد يجب أن يكون متكيفاً مع شروط محدّدة ناتجة عن مميزات محيط العمل المنظور (عتاد ومناهج) . هذا التكييف ، الذي يتم عادة في لغة كبيرة منه بواسطة مصرّف ، يمكن أن يحتاج أولاً - لمساعدة المبرمج .

على سبيل المثال ، فلنذكر بعض الحالات حيث نحتاج إلى بعض التكيف :

- تكيف برنامج مجرد لشروط الموارد بالنسبة لمحيط تشغيل معين .

- إعادة ترميم المناهج الموجودة ، المكتوبة بلغة أخرى ، وإدخالها في مناهج في طور الإنتاج .

- إنشاء برنامج جديد يستغل معطيات قديمة ، مبنية على ناقل بشكل تختلف عن معايير نظام البرمجة المستعمل .

- برمجة أقسام من الأنظمة المتعلقة بالعتاد : إدارة الذاكرة ، إدارة الإدخال - الإخراج ، نواة المزامنة ، الخ .

ولكن ، إستعمال وسائل التكيف تقلّم سيئات جدية ولا يجب أن تستعمل إلا لحاجات ضرورية . هكذا ، فالبرنامج المجرد يعالج المسألة بشكل عام ويشكل سهل الرؤية ، قابل للتطوير وصالح للعمل - لأنه يمكننا أن نقوم بإثباته بسهولة - من برنامج « على القياس أو حسب الطلب » خاص في محيط تنفيذي معين . إضافة لذلك ، فالبرمجة المجردة تؤمن إمكانية النقل ، التدقيق الساكن والانتاجية .

2.1.2 مبدأ فصل الصفات المجردة والمميزة

لأن عمليات التكيف هي ضرورية في بعض الأحيان ، يجب حصر آثارها المؤذية : هذا هو مبدأ الفصل بين الصفات المجردة والمميزة لأحد البرامج ، كما كان ذلك مطبقاً في اللغة LIS .

في البرمجة التناقضية ، فالمرور من دفتر الشروط إلى برنامج العمليات يتم بواسطة مراحل متتالية كما يشير الجدول 1 . كل مرحلة يمكن أن تعتبر كعملية تكيف للسابقة . ومبدأ الفصل يجب أن يطبق بالتحديد : بين الأقسام الخاصة والعامة للزجلة (مرحلة 3 ، جدول 1) ، ولكن أيضاً بين الأوصاف المنطقية وتمثيلها الفيزيائي (مرحلة 4) . البيانات المستخلصة لعمليات التكيف يجب أن تكون مركزية ، واضحة ، سهلة للاخفاء ، للمراجعة أو للتعديل . هكذا ، ففي حالة تغيير نص التشغيل ، أو في حالة إنتاج أمثلة متعددة (مناهج تطبيقية مخصصة لمختلف مراكز الكمبيوتر) ، فقط بيانات التكيف هي للمراجعة .

الامكانيات المقدمة من آدا	مميزات عمليات الوصف الناقية	مراحل
لا يوجد شيء	دفتر شروط مفهوم عملي والشروط المفروضة أو المطلوبة للبرجة والتشغيل	تحليل تقريبي
تركيب إنشائي ممكن للمسائل الثانوية بواسطة رزم . مفهوم دلالي في ملاحظات	مواصفة عملياتية : تعريف دقيق للمفاهيم العملية للمسألة ، تركيب إنشائي في مسائل - ثانوية . عمليا لا يوجد أي شرط في التشغيل	تحليل عملياتي
جميع إمكانيات آدا ما عدا وسائل التكييف المعروضة في الفصل [MR 13]	ماكيت عملياتية أو برنامج مجردة : تطوير الأقسام الخاصة للزرم : إختيار بنية المعطيات والخوارزميات مع أخذ بعض مفاهيم الاستقلال بالاعتبار ، حجم المعطيات ، تردد النداءات للمهام ، لا نحتاج لاية مميزات للمعاد المستعمل .	برجة : تحليل عضوي ، برجة بجودة ، تكييف خوارزمي
- خواص ونصوص - وسائل تكييف مشروحة في الفصل 13 من [MR]	برنامج عملياتي : إعتبار الشروط الخاصة بمفهوم التشغيل . تكييف الماكيت مع الشروط المنظورة . الحصول على برنامج يكفي هذه الشروط ، ولكن قد لا يكون قابلاً للتنقل .	تكييف حسب شروط التشغيل الخاصة

جدول 1 - تطوير ممكن للبرنامج في لغة آدا

ملاحظات :

- الفصل بين الأقسام الخاصة والعامة للزجلة هو واضح في اللغات Alghard, CLU ، وأدا ، وهو أقل جودة في Modula-2 .
- الفصل بين الوصف المنطقي وتمثيله الفيزيائي يستعمل مختلف التقنيات . في LIS وفي Modula-1 ، البنات المذكورة ليست مسموحة إلا في الأقسام النصية الخاصة . في Module 2 ، عندما تكون الزجلة متعلقة بالمكنة الفيزيائية ، فهو يشير إلى ذلك بوضوح بإدخال زجلة خاصة («Module SYSTEM»).

12.1.2 وسائل المساعدة في التكيف

- إنتاج البرامج يتجه ليصبح ، وبالأخص في المعيار الصناعي ، عملية معقدة تجمع بين النشاطات العملية ، التي يقوم بها الأشخاص المختلفين أو فرق العمل . فلنذكر مثلاً ، مواصفة المناهج المطلوب إنشاؤه ، تصور هيكلية ، اختيار الخوارزميات وتركيبات المعطيات ، البرهان ، التوثيق ، محاولات القياس ، التكيف ، الخ .
- كل نشاط من هذه النشاطات يمكن أن يتم بواسطة وسيلة محدّدة ، والإتجاه الحالي في صناعة المناهج هو في محاولة تجميعها في نظام مساعد في البرمجة .
- تنظيم هذه الأنظمة يمكن أن يتم بطريقة مركزية أو موزعة ، مع الفوائد والسيئات المحددة للتركيبة المختارة .

عملية تصور نصوص لمختلف هذه النشاطات في نفس اللغة تؤدي إلى تعقيد اللغة وتعقيد مصرّف اللغة ويجعل تعلمه أصعب . وعلى العكس ، إذا عرضنا لغات ووسائل خاصة لكل فعالية ، فهذا لن يكون ذا فائدة إلا بالنسبة للصناعات الكبيرة التي تستعمل أشخاصاً كثر لنفس العمل (التخصص في العمل) . الحل الوسط الممكن يبدو وكأنه في لغة تركيبية في عدة مستويات متماسكة ولكن اختيارية .

في هذه الحالة يمكن للمصرّف ، للوسيلة ، أو للمستعمل ألا يأخذ إلا ببعض المستويات . هذا المفهوم يفرض غالباً أن تكون المستويات نموذجية ومحدّدة في المساعد المرجعي للغة .

12.1.4 المهام المطلوب تنفيذها

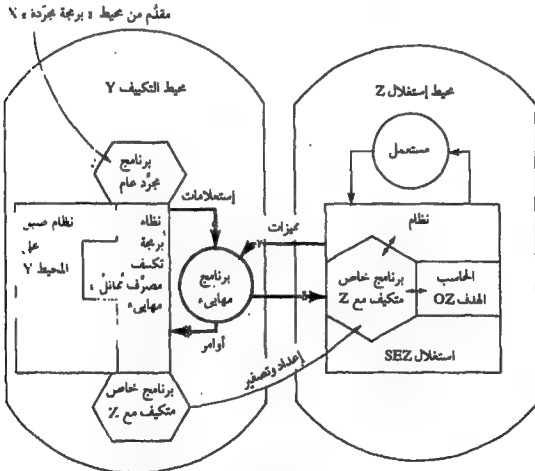
- « المبرمج - المهام » : يجب أن يكون (شكل 1) : - على علم بمميزات محيط برنامجنا خلال إنتاجه أو خلال تشغيله
- قادراً على تنظيم سلوك عام أو سلوك محدّد بالكامل لاجراء التكيف (برنامج مجرد ← برنامج خاص)
- قادراً على تنظيم أفعال لمحيط الاستغلال المنظور

فلنشر إلى إن الحوار بين هذه المحيطات والمبرمج يمكن أن يتم بطريقة تخطائية أو غير ذلك . إن عملية اعتماد ، في لغة للبرمجة ،.. كما هو الحال في لغة أدا وفي أغلب اللغات الحالية .. تنصوص ووسائل لتأمين هذا الحوار هو فعل جدير بالمعارضة : هو الاختيار بين الحوار الجامد بواسطة برنامج مُتدخل .

الشكل (1) يعرض العلاقات السابقة ، في حالة إنتاج متعدد الأهداف حيث محيطات البرمجة (X) والتشغيل (Z) هي محدثة . مرحلة التكيف يمكن أن تتم على X أو على Z أو على أية محيط ما عدا Y .

12.2 - مبادئ لغة آدا

تقلم لغة آدا وسائل لإقامة كل من العلاقات المدروسة سابقاً (الأسهم $\alpha, \beta, \gamma, \delta$) .



شكل 1 : العلاقات بين المبرمج ومحيطات التكيف والاستغلال .

12.2.1 العلاقات بين المبرمج ومحيط البرمجة (السهم ، ٧ ، ٨)

يجب أن يكون المبرمج قادراً على معرفة القرارات التي يأخذها المصِّرف (مثلاً تمثيل للمعطيات) أو التي يأخذها زملاؤه (مثلاً قيمة إحدى المتغيرات) كي يستطيع أن يقوم بنفسه بتقديم نصائح عامة أو دقيقة لتوليد الكود إلى المصِّرف.

12.1.2 12. معلومات المبرمج

حالياً ، هناك قليل من اللغات التي تفرض في نحوها تركيبات وإنشاءات تسهل تنظيم التوثيق ومراقبة معلومات المبرمجين . كل معلومة غير مفيدة مباشرة للترجمة أو لعمليات المصِّرف في المراقبة ، توضع في الملاحظات . ولكن هل من الواجب أن تسهل اللغة المهمة وذلك بعرضها وسائل لتركيب وتمييز مختلف أنواع العمليات والوثائق في نحوها .

لغة آدا لا تقدم جديداً في هذا الحقل . الشيء الأساسي من المعلومات التوثيقية يُوضع في الملاحظات . ولكن التركيبة الزجالية لبرامج آدا - وبشكل خاص الفصل بين الأقسام العامة والخاصة من الرزمة - تقدم بعض الوسائل في التركيب وتنظيم الوثائق .

الإمكانات المقدمة بواسطة الخاصيات هي أكثر غناءً . هذا المفهوم جرى تطويره في لغة آدا . وباستطاعة المبرمج أن يعالج بعض الخصائص التي تعتبر غلبة في الوضع الطبيعي (مثلاً حجم المتحولة) وذلك من أجل تثبيت أحد التصاريح أو لتحديد عملية تمثيل معينة .

12.2.1.2 معلومات المصِّرف

يوجد في لغة آدا عدة وسائل للتأثير على عمل المصِّرف . أغلب هذه الوسائل يتعلق بتوليد الكود .

- الدلائل تقدم نصائح عامة صالحة لقسم من النص : جعل الوقت أو المساحة من الذاكرة مثل ، الخ . بعض الدلائل الأخرى تستخدم للدناء إلى المكتبة أو لصناعة النسخ المطبوعة .

- مواصفات التمثيل لمُجبر المصِّرف على تمثيل ، إما تركيبة معطيات في نسق مفروض جزئياً أو كلياً ، أو معالجة في لغة مكنة معينة (تدلخل النصوص في شبه - لغة تاويل) .

- الوصلات مع اللغات الأخرى . يمكن أن يكون مصِّرف آدا مجبراً على توليد كود متوافق مع اعتبارات لغات أخرى . وذلك عند دعوة برنامج - ثانوي (Pragma INTERFACE) . بإمكاننا إذاً ترميم أقسام من البرنامج مكتوبة بلغات أخرى .

- إلغاء عمليات التحكم المولدة بواسطة المصِّرف . بعض عمليات التحكم من الأنواع

المنشأة عند التصريف أو التنفيذ هي إلزامية عندما يتم إعتبار موضوع معين بأشكال مختلفة . (مثلاً ، عند كتابة غُصَص للذاكرة أو لإجراء تحويل) . في أدا ، من الممكن إلغاء بعض عمليات التحكم هذه (pragma SUPERSS) ، أو إستعمال رزمة نموذجية (- UNCHECKED - CONVERSION) .

12.2.2 العلاقات بين المبرمج ومحيط التنفيذ (الأسهم * ، α) .
يجب أن يستطيع المبرمج معرفة مميزات محيط التنفيذ المنظور ، أو إعطاء أوامر لهذا المحيط عند الاستغلال ، وبواسطة برنامجهِ . اللغة كويول كانت تعرض في نحوها التركيبية (ENVIRONMENT DIVISION) لتوضيح هذه العلاقات . ولكن أغلب اللغات كانت تهمل هذا التوضيح .

12.2.2.1 معلومات المبرمج
من المعلوم أن هذه العلاقات تتعلق بالطرق المعتمدة بواسطة نظام البرمجة . وهناك عدة حالات يمكن أن نتقدم :

أ - لغة البرمجة نفسها هي موجهة ، على الأقل في بعض أقسامها ، نحو مكنة معينة مستهدفة (إختيار 1-Modula أو 2-Modula) .

ب - اللغة هي مستقلة عن أية مكنة ، ولكن مصرفاتها يمكن أن تكون متخصصة لمكنات خاصة .

ج - تقسم المصرفات إلى قسمين ، قد ينفذان في محيطات مختلفة . القسم الأول يقوم كوداً لمكنة مجردة (لغة بسيطة) ولكن متكيفة (بالترجمة أو بالتأويل) مع محيط معين بواسطة القسم الثاني الذي وحده يتعلق بها . نلاحظ إن هذا التقسيم يسمح إذاً للغة المحللة في قسمها الأول بأن ينزع منها جميع نصوص التكييف ، لأن هذه الأخيرة تتعلق فقط بالقسم الثاني .

تعريف لغة آدا لا يسمح أبداً بمعرفة ما إذا كانت المصرفات هي من النوع ب أو ج المذكورين . هناك بعض الإنشاءات من النوع ب : نصوص تكييف خاصة لمكنة معينة ، وجود في كل مساعد - مرجعي للملاحق C و I التي تناسب محيطات التنفيذ والبرمجة . ولكن وجود الزريعة Pragma SYSTEM التي تشير للمصرف الشبه مع المكنة - الهدف ليس لها أية فائدة بالنسبة للمصرفات من النوع ج) .

باستطاعة المبرمج بلغة آدا - أن يكون على معرفة بمميزات محيط الاستغلال ، أو التشغيل ، باستشارة نص التوثيق والمراجع : وثائق النظام المستهدف ، الملاحق C و I للمساعد Ada . وبشكل خاص ، الرزمة النموذجية SYSTEM التي ترسل المميزات الخاصة للمكنة الهدف .

12.2.2.2 معلومات المُغْذ

كما في جميع اللغات ، فإن محيط التنفيذ يستقبل الأوامر الآتية من البرامج ، بشكل مباشر أو غير مباشر .

في لغة آدا فإن هذه الإمكانيات تزداد غناءً بواسطة الرزم أو المهام التي تستطيع إحتواء عتاد مختلف . هكذا ، فكل مكتبة مستهدفة يمكن أن ينظر إليها المبرمجين وكأنها رزمة آدا (تستعمل هذه الخصوصية للدخال - الإخراج) .

12.3 - الخاصيات والرزمة SYSTEM

المراجع : 3.3, 3.5, 3.6.2, 3.7.2, 4.1.4, 13.2, 13.3, 13.7

ملحق A, C, F

Me: 14.6.2

12.3.1 مبادئ

ليست للخاصيات (attributs) أي دور مميز في لغة آدا . فقط المساعد [MRA] هو الذي يعطيها تعريفاً عاماً : « الخاصية هي مُعرِّزة مُعدَّلة مسبقاً كوحدة ومعنية بواسطة اسم » . يبدو أن النقاط الوحيدة المشتركة بين مختلف الخاصيات هي الشكل النحوي الذي يعينها [MR 4.1.4] وصفتها مُعدَّلة بشكل مسبق . يمكن أن تكون نموذجية [MRA ملحق A] أو خاصة بمصرف معين [MRA ملحق F ، فقرة 2] . أما بالنسبة للرزمة المُعدَّلة SYSTEM ، فهي تنقل مُميزات معينة خاصة بالمحيط المنظور .

تختلف الوحدات التي تستطيع الحصول على خاصيات معينة : موضوع ، نوع ، برنامج ثانوي أو مهمة ، طبيعة الخاصية . يمكن أن تكون قيمة ، دالة أو نوعاً (ثانوي) . ولكي يتم استعمال الخاصية بشكل صحيح يجب معرفة نوعها (قيمتها) أو رأسها (دالة) . هذه المعلومات هي موضحة في MRA ملحق A .

مثال رقم 1

COULEUR*FIRST

- القيمة الأصفر للنوع COULEUR

SYSTEM-MIN-INT

- القيمة الدنيا الصحيحة للمكتبة المستهدفة

MATRICE*LAST (2)

- نداء الدالة MATRICE*LAST الذي يعيد الحلق

الأعلى للبعد الثاني للمصفوفة

تخضع الخاصيات التي تُثَّمل نوعاً معيناً إلى قاعدة خاصة في الاستعمال : لا يمكن استعمالها إلا لتحيين خاصية أخرى .

مثال رقم 2

T*BASE*FIRSTT

- القيمة الأولى للنوع الاساسي للنوع الثانوي

فائدة الخاصية تتعلق بنوع الميزة المعنية . هناك ثلاث فئات منها .

12.3.1.1 الخصائص المجردة

يتعلق ذلك بالمعلومات المجردة المرتبطة بنوع أو بنوع - ثانوي معين .

● يختلف	BASE
● لا اتجاهي	FIRST, LAST
● متجزئاً	POS, VAL, PRED, SUCC, IMAGE, VALUE, WIDTH
● وفاصلة ثنائية	DELTA, BITS, LARGE, FORE, AFTER, SMALL
● وفاصلة متحركة	DIGITS, MANTISSA, EMAX, SMALL, LARGE, EPSILON, MODEL (SMALL, EMAX, LARGE)
● جديلي	FIRST, LAST, LENGTH, RANGE
● فقرة مع محيز	CONSTRAINED

مثال رقم 3

تعرف الرزمة نوعاً مرقياً

type COULEUR is (BLANC, ROUGE, ..., BLEU, NOIR) .

هناك رزمة Q تستعمل النوع COULEUR ، ولكنها تحمل القيم التي تؤلفه .
ويمكن بشكل مجرد ، بسبب وجود الخاصيات ، استعمال جميع خصائص COULEUR ،
كما يؤمن تطويراً إيجابياً .

C : COULEUR ;

```
for C in COULEUR'FIRST .. COULEUR'PRED (COULEUR'LAST)
loop -- C تأخذ القيم BLANC .. BLEU
    الحالة الإستثنائية CONSTRAINT_ERROR سيتم
    إطلاعها إذا لم تحصل COULEUR على أية قيمة --
end loop
```

12.3.1.2 الخصائص الخاصة والرزمة SYSTEM

هذه الخاصيات هي عبارة عن معلومات خاصة مرفقة بالوحدة

- موضوع أو برنامج ثانوي ADDRESS
- موضوع ، نوع (- ثانوي) ، مهمة SIZE
- نوع فاصلة ثابتة MACHINE-ROUNDS
- نوع فاصلة متحركة
- MACHINE-(RADIX, MANTISSA, EMAX, EMIN, ROUNDS, OVERFLOWS)
- نوع فقرة POSITION, FIRST-BIT, LAST-BIT

- نوع مؤشر STORAGE-SIZE
- نوع أو موضوع مهمة STORAGE-SIZE

الرمزة SYSTEM [MRA 13.7] تقم نفس النوع من المعلومات ولكن للمكنة الهدف .

استعمال هذا النوع من المعلومات ، على عكس الخاصيات المجردة ، يؤدي إلى سيطرة خطية بهبوط مستوى التجرد في البرنامج ، وذلك بجعله متعلق بمحيط البرمجة أو التنفيذ ، هذه الخاصيات مفيدة لكتابة بعض أقسام المناهج «systemes» . ومن جهة أخرى تستخدم لمواصفة التمثيل .

مثال رقم 4

For MOT-ETAT-PROGRAMME'SIZE use 2 * SYSTEM. STORAGE-UNIT;

- يجعل المصنف يستعمل كلمتين من المكتبة الهدف لتمثيل التحولات من نوع MOT-ETAT-PROGRAMME

NB-REGISTRES : INTEGER : = ** INSTRUCTION. REG'SIZE

.. إعداد متحولة مع قيمة ثابتة للرمز التنفيذ : عدد مرادفات للمكتبة .

12.3.1.3 خاصيات أخرى

لا يوجد للخاصيات أي شيء مشترك مع الفئات السابقة :

.. مهمة : TERMINATED, COMPLETED أنظر 8.4.3

.. مدخل : COUNT أنظر 8.3.1 .

يختص ذلك بثلاث خاصيات تجعل للمعلومات عن الحالة معروفة ومتحولة عند التنفيذ .

12.3.2 التقييم

12.3.2.1 النقاط الإيجابية

المعلومات المقدمة بواسطة الخاصيات جميعها مفيدة وتسمح بتثبيت وتكييف شديد من الصعب أن نحصل عليه بواسطة اللغات الأخرى .

12.3.2.2 النقاط القابلة للنقاش .

يبدو مفهوم الخاصية وكأنه عام لأنه يعني «ميزات ذات طبيعة وإستعمال مختلف» . هكذا ، فهو لا يمثل سوى ميكانيكية تحديد الخصائص ، من مجموعة ممكنة كالتعبير المؤشر للوحدات المنقولة بواسطة ريمة ، والتعبير العملياني .

مؤلفو آدا يُبررون هذا الاختيار والمفهوم المحفوظ ، لتفادي التنازع بالأسماء مع
الأسماء المحلدة بواسطة المبرمجين .

هكذا ، فهذا التعبير الوحيد لن يكون الأفضل تكييفاً بالنسبة للفئات الثلاث التي
قمنا بتمييزها سابقاً .

أ - الخاصيات المجردة

بما أن استعمالها ليس له أية سيئة بالنسبة لامكانية النقل ، فلا يجب حصر
إستعمالها . هكذا فهل كان من الممكن تمثيل الخدمات التي تقدمها هذه الخاصيات بواسطة
دوال محددة مسبقاً ؟ . هل كان من الواجب حصر شروط إستعمال الخاصية BASE ، بينما
لا يوجد شروط بالنسبة للخاصية RANGE التي تختص أيضاً بالنوع ؟

مثال رقم 5

```

declare
INDICE : T'RANGE ;
B1 : ST'BASE ;
type BASE_ST is range ST'BASE'FIRST .. ST'BASE'LAST ;
B2 : BASE_ST ;
end ;

```

- صيغة نموذج 80
- تسمح بعبور الجداول T
- غير مسموح
- مسموح ولكن إنشاء نوع مشتق
- مسموح

فإذا لم تكن من دعاة صفة الاسقاط orthogonalité بأي ثمن - التي تسمح بتبسيط
لغة آدا- لكان من الممكن إعتبار الحل المقدم كافياً .

ب - الخاصيات الخاصة والرمز SYSTEM

في هذه المرة يؤدي إستعمال هذه المميزات إلى سيئات جدية بالنسبة لامكانية النقل ،
ونأسف بأن تكون التعابير المفروضة غير خاضعة لقيود في الاستعمال .

هكذا ، فاستعمال الرمز SYSTEM أو الخاصية ADDRESS التي تجعل قيمة من
نوع SYSTEM-ADDRESS يحتاج إلى استعمال الجملة with SYSTEM في رأس كل وحدة
تصريف مُستعملة . وهذا هو تطور في [MRA] بالنسبة للصيغة السابقة [MR] .

مثال رقم 6

```

with SYSTEM ; use SYSTEM ;
package body Exemple_6 is

```

- وحدة تصريف
- المتاليان التاليان يبرهnan ما يستطيع أن يفعله المبرمج المتاد على البرمجة بلغة المؤول ، لكنه سيقوم بلفت
إنتباه رئيس فريق عمله بسبب الجملة With SYSTEM .

```

TABLE : array (INDEX) of ENTREE ;
I : INDEX ;

```

LG_TABLE : constant ADDRESS := I'ADDRESS - TABLE'ADDRESS ;

- مغلوط: هذا هو مقبول بواسطة اللغة ولكنه سيكون صحيحاً بالنسبة لبعض المصنفات وغلط بالنسبة للآخرى .

type VECTEUR is array (1..NBELEM) of FLOAT ;

LG_VECTEUR : constant INTEGER := NBELEM*SYSTEM.STORAGE_UNIT

أيضاً مغلوط : هذا لن يكون مغلوطاً إلا بالنسبة للمكنات التي يمثل بها FLOAT بواسطة كلمة . هنا

end Exemple_6 ;

كان يكفي بأن نكتب VECTEUR'SIZE .

بالنسبة لجميع الخاصيات الخاصة ، كان من الواجب جعل إستعمالها واضحاً ، وحصر شروط إستعمالها ، باعتماد أولية نقل وإدخال الوحدات الواقعة في الرزمة SYSTEM ، أو بتمييز الوحدات التي تحتوي على هذه الخاصيات بواسطة صفة معينة (مثلاً concrete) .

ج - الخاصيات الأخرى

هذه الخاصيات لا تمثل أية أخطار خاصة ، ولكن من الملفت إستعمال تعابيرها للدلالة على قيم الحالة . لماذا لا نقوم بالإشارة إلى قيمة إحدى المتحولات بهذه الطريقة ؟ السبب في ذلك يأتي أيضاً من عدم وجود صفة الأسقاط (orthogonalité) في لغة أدا : الدالة المحددة مسبقاً - التي تبدلنا وكأنها متكيفة بشكل أفضل - يمكن أن تحصل على مهمة ، ولكن ليس على مدخل كمعامل وإذا كان السبب الفعلي هو في عدم « تدنيس » الأسماء التي يضعها المبرمج بالأسماء من نوع COUNT وTERMINATED ، لكان من الواجب إستعمال تعبير الخاصية للدالة الوحيدة المحددة مسبقاً : ABS .

12.4 - النرائع

المراجع

MRA :2.8, 4.8, 6.3, 9.8, 10.3, 11.7, 13.1, 13.7, 13.9 ; Annexes B, F.

ME : 14.6.1, 14.6.3.

12.4.1 مبادئ

النرائع تشكل الوسيلة الممنوحة للمبرمج والتي تسمح له بأن يأمر نظام البرمجة بالتصرف بسلوك عام حسب البرنامج الخاضع له . القسم المختص من النص (مدى الذرية) والمكان الذي من الممكن أن توضع فيه يتعلق بالذرية . هذه القواعد ستكون صعبة للوصف في النحو ، لهذا تعتبر الذرية بشكل عام « كوحدة نحوية » . [2.8] .

بوضع LIST وINCLUDE جانباً ، نستطيع القول إن جميع الدلائل تتعلق بتوليد الكود : فهي إذاً عبارة عن وسائل للتكيف . من الممكن أيضاً اعتبارها مضافة للخصائص أو للوحدات المنقولة بواسطة الرزمة SYSTEM .

مثال رقم 7

pragma SYSTEM (X) ↔ ثابتة SYSTEM.NAME

هذا التماكس هو أقل منه في [GREEN] أو [MR] .

pragma OPTIMIZE (TIME) ↔ خاصية OPTION TIME - قبل [MRA]
pragma PRIORITY ↔ attribut PRIORITY - قبل [MRA]

بإمكاننا ، كما فعلنا بالنسبة للخصائص ، تمييز عدة فئات من الدلائل .

يمكن للدلائل أن تكون مغلقة مسبقاً [MRA ملحق B] أو خاصة بمصرف [ملحق MRA F] . إذا لم يكن بإمكان المصرف أن يقوم بتنفيذ بعض الدلائل فيجب الإشارة إليها [MRA ملحق F] .

12.4.2 أمثلة على الاستعمال

12.4.2.1 تصريف مشروط

مثال رقم 8

pragma SYSTEM (PDP11_23) ;
pragma MEMORY_SIZE (1000) ;
package body Exemple_8 is -- وحدة التصريف / الأولى :
 - في هذه الوحدة ، سيتم إنتاج الكود بالنسبة للمكتبة PDP11 - 23 والمصرف سيكون على علم بأن هذه الرزمة لا يجب أن تتجاوز 1000 كلمة .

```

=====
case SYSTEM.NAME is
  when PDP11_23 =>
    declare L : constant INTEGER := SYSTEM.MEMORY_SIZE ;
    begin
      ...
    end ;
  when MINI_6 =>
    ...
end case ;
=====
end Exemple_8 ;

```

هذا التثبيت هو شديد الفعالية طالما إن SYSTEM.NAME هي ثابتة ومعروفة من قبل المصرف : يمكن ألا يولد إلا الكود المناسب للحالة للملائمة . يمكن لرزمة أخرى أن

تعرف التمثيل الفيزيائي للفقرات المصريح عنها في الرزمة السابقة وذلك بسبب وجود الخاصيات POSITION ، FIRSTBIT ، الخ .

12.4.2.2 رص المعطيات

لنفترض أن الحاسب المستهدف هو حاسب بكلمة ويعدد صحيح بطول 32 بتة ، مع سهولة لبولوج سمات بطول 8 بتات . المثل الثاني يبرهن إستعمال الذريعة PACK التي تقلل المساحة غير المستعملة - بين عناصر الفقرة (gaps) (أو الفجوات) ، دون رص العناصر نفسها .

مثال رقم 9

```

declare
type ENUM is (V1, V2, ..., V8);
type TABENUM is array (1..2) of ENUM;
type ARTICLE is
  record
    A : array (1..5) of CHAR;
    B : INTEGER;
    C : TABENUM;
  end record;
pragma PACK (ARTICLE);
end;

```

- مكوّن على ثلاث بتات

- عملاً 5 بايتات

- 4 بايتات

- عملاً 2 بايت

- إستعمال الخاصية T هو ممنوع هنا بسبب الذريعة التالية :

- تتمتع الخاصيات التالية بالقيم المحددة :

-- T'SIZE = 96, T'B'POSITION = 2, T'C'POSITION = 1

- فقط هناك بايت واحدة مفقودة في نهاية الكلمة الثانية .

12.4.2.3 مساحة الذاكرة المثل

على عكس PACK ، فإن الذريعة OPTIMIZE لها فعل غير واضح تماماً ، ومداها يمتد إلى كامل الوحدة حيث هي موضوعة . يجب أن يكون لها تأثير على التصريحات التي تتبعها :

مثال رقم 10

```

package body Exemple_10 is
  T : array (1..X) of Y;
  U : array (1..T'SIZE) of BOOLEAN;
  Z : INTEGER := U'ADDRESS + T'SIZE;
pragma OPTIMIZE (SPACE);
end Exemple_10;

```

- نسخة H0 ، نسخة H2 ؟

- عدد التصرّيات ؟

12.4.2.4 ماكرو - تعليمات (التعليمات الكبرية)

إستعمال الذريعة `INLINE` يسمح باعتبار البرامج الثانوية كتعليمات كبرية ، مما يسمح بتفادي ثمن نداثها . المثال التالي يدل على طريقة إستعمال ممكنة لعمليات البلوغ السريعة إلى عناصر من المصفوفات .

مثال رقم 11

```
generic
type ELEM is private ;
type IND_L is range <> ; - مؤشر سطر -
type IND_C is range <> ; - مؤشر عمود -
package GESTION_MATRICES is
type MATRICE_CREUSE is private ;
function ACCES_GEN (I : IND_L, J : IND_C) return ELEM ;
function POSIT_LIG (I : IND_L) return ELEM ;
function ACCES_LIG return ELEM ;
function ACCES_COL return ELEM ;
-----
pragma INLINE (ACCES_LIG, ACCES_COL) ;
end GESTION_MATRICES ;
```

فقط عمليات البلوغ على السطر أو العمود ، والمفترض أن تكون سريعة جداً وقليلة الحجم ، تعمل كتعليمات كبرية .

12.4.3 تقييم

12.4.3.1 النقاط الإيجابية

بالمقارنة مع اللغات الأخرى ، فالتفكير بإدخال تعابير خاصة بالنسبة للأوامر التي يصدرها المبرمج إلى المصنف هو عبارة عن تقدم كبير : فلفة باسكال لا تعرض سوى الذريعة `PACKED` ، ولكن بعض المصنفات تعرف ملاحظات خاصة مثلاً

الأمثلة السابقة تدل على أن لغة آدا تمتاز بإمكانيات تثبيت وتكييف قريبة من إمكانيات المأول - الكبري (`MACRO ASSEMBLER`) ، مع المحافظة على جميع فوائد مميزات اللغة ذات المستوى العالي .

في الصيغ الجديدة تخضع النرائع (مثلاً `SYSTEM`) لقواعد إستعمال دقيقة ، لأنها يجب أن تظهر على رأس التصريف : وهذا هو جيد ! .

12.4.3.2 النقاط القابلة للنقاش

كما وبالنسبة للخاصيات ، فإن للنرائع فوائد مختلفة وهي لا تؤلف التعبير الوحيد لتأمين التكيف ، لأنه يوجد أيضاً مواصفات التمثيل .

التعقيدات الناتجة عن قواعد المدى وموقع بعض الذرائع يجعلنا نتألم إلى وجودها في اللغة . فنحن أمام كشلة شبيهة بمشكلة تنقيح النصوص : هل يجب خلط الأوامر إلى النص أو استعمال لغة أوامر خارجية ؟ (بعض متفحات النصوص تستعمل لوحات ملامس مختلفة للنص وللأوامر) . الحل الثاني يبدو لنا وكأنه الأفضل ، ولكنه يفترض استعمال وسائل عتاد مخصصة لتعريف مدى الأمر ، ويجب المحافظة على أثر أوامر التكيف .

12.5 مواصفات تمثيل المعطيات

المراجع :

MRA : 13.1, 13.2, 13.3, 13.4, 13.5, 13.6
ME : 14.2, 14.3, 14.4, 14.5

12.5.1 - المبادئ

يتميز كل موضوع (متحولة أو ثابتة) بصفات منطقية وفيزيائية (مجردة وخاصة) الصفات المنطقية توضح الاستعمال الممكن للموضوع : قيم ، مؤثرات وبرامج - ثانوية قابلة للتطبيق . الصفات الفيزيائية تتناسب مع التمثيل في الذاكرة ومع المميزات التكنولوجية للعتاد المعتمد . بشكل عام ، تُدعى نوعاً مجرداً فئة - تعادل المواضيع التي تتمتع بنفس الخصائص المنطقية ؛ يمكن أيضاً أن تدعى نوعاً خاصاً تلك التي تتناسب مع الصفات الفيزيائية .

في آدا ، هذان التعبيران عن النوع هما متوافقان : جميع المواضيع من نفس النوع لهم نفس الخصائص المنطقية والفيزيائية . ينتج عن ذلك إن مواصفات التمثيل تنطبق على الأنواع ، وليس على المواضيع المعزولة (ما عدا بالنسبة للعناوين) .

من الواضح ، إن المواضيع من النوع المجرد ولكن من أنواع خاصة مختلفة تشابه كثيراً : تنطبق عليها نفس الخوارزميات المجردة . وأيضاً بنفس البرامج - الثانوية التي تستعملها كعمليات . لتفادي عملية إعادة برمجة هذه البرامج الثانوية ، ولتسهيل تبادل القيم بين هذه المواضيع (تغيير التمثيل) ، نقدم لغة آدا تصورات من النوع المشتق والبرامج - الثانوية المشتقة .

مثال رقم 12

declare

- لنفترض نوعاً منطقياً JOUR مصححاً عنه كالآتي :

type JOUR is (LUN, MAR, MER, JEU, VEN, SAM, DIM) ;

- من الممكن تعريف الاشتقاقات من JOUR ، بالمحافظة على نفس الخصائص المنطقية ، ولكن بخصائص

فيزيائية مختلفة :

type JOUR2 is new JOUR ;

type JOUR3 is new JOUR ;

for JOUR use - تعريف توكيد للقيم المجردة

```

(LUN => 1, MAR => 2, MER => 3, JEU => 4, VEN => 5,
 SAM => 6, DIM => 7);
for JOUR2 use -- تعريف تكوين آخر --
(LUN => 3, MAR => 4, MER => 6, JEU => 10, VEN => 11,
 SAM => 15, DIM => 20);
for JOUR3*SIZE use 3 ; -- استعمال ثلاث بنات من أجل التمثيل --
    بالقابل ، لا يمكننا تصريف الأنواع المشتقة التالية --
type DAY is new JOUR ;
type JOUR_MUSULMAN is new JOUR ;
for DAY use (MON => 1, TUE => 2 ... ) ;
-- تغيير القيم المجردة --
for JOUR_MUSULMAN use (SAM => 1, DIM => 2 ... ) ;
-- تغيير الأمر --
- يجب إنشاء ، أنواع جديدة ، إذا أردنا إستغلال البرامج - الثانية المشتركة بالأنواع DAY و JOUR ،
JOUR-MUSULMAN ، وهذا لا يتم إلا بواسطة برامج ثانوية نوعية عامة تتمتع بأحد هذه الأنواع
كمشغول للمعمولة .
end ;

```

12.5.1.1 مواصفات التمثيل

هي عبارة عن الإشارات المقدمة إلى المصرف لتمثيل أنواع المعلومات والمواضيع
 أكانت مشتقة أو غير مشتقة . وهي إلزامية للمصرف ، ولكنها يمكن أن تكون جزئية أو
 كاملة . في غياب هذه التأشيريات ، يقرر المصرف ما يجب عمله حسب رأيه . وفي آدا ،
 تسمح هذه التأشيريات :

- تحديد حجم الذاكرة المطلوب تخصيصها للمعلومات والمواضيع من نوع معين ، أو
 لمجموعة منها ، أو لمهمة .
- تكوين القيم المجردة من نوع مرقم .
- التمثيل الخاص لفقرة .
- عنوان مكان وجود الموضوع أو البرنامج الثانوي .

فلذلك هنا إن بعض اللوائح (مثلاً PACK) تلعب دور المواصفات غير الكاملة
 والاختيارية .

هذه المواصفات يمكن أن تظهر في أي مكان من القسم الوصفي ولكن في نفس
 القسم الذي يحتوي على الوحدات الموصوفة ، وقبل أي استعمال للخاصيات أو للبرامج
 الثانوية المشتقة . عدة مواصفات يمكن أن تنطبق على نفس النوع إذا لم تكن متناقضة : مثلاً
 مواصفات الطول وتكوين النوع المرقم . وفي النهاية ، فإن استعمال مواصفات التمثيل
 للأنواع المشتقة يجب أن يخضع لعدة قواعد معقدة [MRA 3.4, 13.1] .

مثال رقم 13

```

declare
  type JOUR is (LUN, ...);
  function MODULO_JOUR (J : in JOUR ; I : in INTEGER) return JOUR ;
  type JOUR2 is new JOUR ;
  type JOUR3 is new JOUR ;
  for JOUR3'SIZE use 3 ;
  for JOUR3 use (LUN => 1, ... ;
-- مسموح [ MRA 13.1 ]
-- غير مسموح بسبب وجود الدالة المشتقة
MODULO_JOUR
-- أنظر [ MRA 13.1 ]

begin
-----
declare
  type JOUR4 is new JOUR ;
  for JOUR4'SIZE use
-- غير مسموح ، يجب وضعه في نفس القسم
-- الوصفي الموضوع فيه JOUR [ MRA 13.1 ]

-----
end ;
-----
declare
  type PJ is access JOUR ;
  type PJ2 is new PJ ;
  for PJ'STORAGE_SIZE use
-- حجم الذاكرة للمجموعة JOUR
2*K*MOTS ;
  for PJ2'STORAGE_SIZE use
-- غير مسموح لأن PJ2 تقسم نفس المساحة
1*K*MOTS ;
-- مثل [ MRA 13.1 ]

-----
end ;
end ;

```

12.5.1.2 تغيير التمثيل

في لغة آدا ، يكون تبادل القيم بين نوعين مجردين متشابهين ، ولكن يتمثل مختلف ، ممكناً بسبب وجود دوال تحويل ينتجها المصروف ، حسب الطلب . في هذه الحالة ، فإن القيم « تغير التمثيل » .

مثال رقم 14

متابعة المثال رقم 13 يمكننا الكتابة :

```

type JOUR1 is new JOUR ;
J : JOUR ; J1 : JOUR1 ; J2 : JOUR2 ; J3 : JOUR3 ; D : DAY ;
-- مشتق مع نفس التمثيل
-----
begin
  J1 := JOUR1 (J) ; J := JOUR (J1) ; -- كلفة صفر عند التحويل
  J3 := JOUR3 (J) ; -- كلفة ضئيلة عند التحويل : (تضاعف)
  J2 := JOUR2 (J) ; -- كلفة أكبر ( جدول التحويل )
  D := DAY (J) ; -- يكتبها المبرمج
end ;

```

دوال التحويل مستعملة بواسطة المصرف لإنتاج الدوال المشتقة [MRA 3.4] .

12.5.2 أمثلة على الإستعمال

12.5.2.1 حجم المتحولات

مثال رقم 15

declare

-- نفترض أن الكلمة الهدف هي بكلمة من 16 بتة .

type COULEUR is (BLANC, JAUNE, ORANGE, ROUGE, VERT,
BLEU, VIOLET, NOIR);

type COULEUR2 is new COULEUR;

K : constant := 16;

type D1 is array (1 .. K) **of** COULEUR;

type D2 is array (1 .. K) **of** COULEUR2;

for COULEUR'SIZE **use** 3; -- ثلاث بتات لكل متغيرة نوع

for COULEUR2'SIZE **use** 8;

for D1'SIZE **use** K * COULEUR'SIZE; -- 48 بتة ولكن مسموح : تناقض --

-- يفترض عناصر رابعة على كلمات متتالية --

-- قد ترفضه بعض المصنفات [MRA الملحق F] --

for D2'SIZE **use** 3 * SYSTEM.STORAGE_UNIT;

-- 48 بتة ولكن غير مشروع : تناقض --

مع المواصفة السابقة لـ COULEUR

pragma PACK (D2);

يمكن في مكان المواصفة السابقة

end;

12.5.2.2 حجم المساحة المخصصة للمجموعة

مثال رقم 16

declare

type PERSONNE;

type PARENT is access PERSONNE;

type PERSONNE is

record

NOM : ALPHA;

NAISSANCE : DATE;

PRED, SUCC : PARENT;

end;

for PARENT'SORAGE_SIZE **use** -- نحو 1000 شخص --

1000 * PERSONNE'SIZE; -- عدد البتات --

end;

هذه الإمكانية تسمح بحفظ ساكن لكان من الذاكرة لمخصص خاص بالمواضيع من نوع معين . الحفظ يتم عند تصميم التصريح ، وربما على مكس التنفيد . الفائدة هنا تأتي عندما نعرف ، قبل التصميم ، العدد الأقصى للأمثلة السابقة أو التي نرغب فيها بمراقبة

المساحة من الذاكرة التي تنظّم ديناميكياً . إذا كانت الأمثلة لا تتغير ، فالتخصيص أو عدم التخصيص الواضح (أنظر 12.7.1.1) هو فعال . في المثال ، لا نستطيع أن نعرف بالضبط عدد العناصر المخصصة ، لأنه يجب حساب المتحولات الضرورية لإدارة المخصص (رؤوس اللوائح ، مثلاً) .

12.5.2.3 حجم المساحة المخصصة للنوع مهمة

عدد البتات `for PILOTE_IMPRIMANTE STORAGE_SIZE use 8 * PAGE ;`

هذه المواصفة ، المقيّمة عند تصميم المهمة ، تدل على الحجم الأقصى للذاكرة الضروري لكس التنفيد . ينصح بتقديمها في كل مرة يمكن أن تكون فيها معروفة ، لأنها تبسّط إدارة الذاكرة وتزيد من سرعة التنفيد .

12.5.2.4 Delta الفعالة لعدد بفاصلة ثابتة

مثال رقم 17

```
declare
type DEGREE is delta 0.1 range - 360.0 .. 360.0 ;
for DEGREE'SMALL use 360.0/2 ** (SYSTEM.STORAGE_UNIT - 1) ;
- يأخذ ، للقسم الكسري ، كامل مكان كلمة غير مشغولة
- بالقسم الصحيح .
end ;
```

هذا النوع من المواصفات يسمح بمراقبة التمثيل الخاص لعدد بفاصلة ثابتة ، معتمدة لأن تكون متكيفة مع الخصائص المجردة ؛ يجب أن يكون :

$$SMALL \leq DELTA$$

12.5.2.5 توكيد الأنواع المرقّمة

مثال رقم 18

```
declare
type CODE_INTRUC is (ADD, SUB, MUL, DIV, LD, ST, ... ) ;
type LETTRE is ('A','B','C', ..., 'Z') ;
for CODE_INTRUC use
( ADD => 1, SUB => 2, MUL => 7, DIV => 8, ... ) ;
- Code EBCDIC
for LETTRE use
( 'A' => 16 # C1 # , ..., 'T' => 16 # C9 # ,
  'J' => 16 # D1 # , ..., 'R' => 16 # D9 # ,
  'S' => 16 # E1 # , ..., 'Z' => 16 # E8 # ) ;
end ;
```

من النادر أن يقوم المبرمج بفرض كود معين لأنواعه المرقمة . هذا قد يحصل عندما يكون مفروضاً بواسطة المحيط الخارجي للتنفيذ . حتى في هذه الحالة ، هناك تناوب : إن يتم تطابق الكود الداخلي والخارجي ، أو تفكيكها باستعمال دوال للتحويل . أفضل اختيار للتنفيذ هو معقد لأنه يتعلق :

- بأهمية الحساب الداخلي المقارن مع التبادل مع المحيط .

- بطبيعة الكود المفروض ، وبشكل خاص عدد وحجم « الفجوات » .

- طبيعة المعالجة التي تستعمل الكود الداخلي : تكرار ، تأثير ، اختيار ، تخصيص ، الخ ..

إن وجود الفجوات يُعقد عمليات ترجمة المصدر الذي يجب أن يستعمل لوائح مترابطة للتكويد .

12.5.2.6 تمثيل الفقرات

مثال رقم 19

declare

- نفترض أن المكتبة الهدف هي بكلمة بطول 16 بت

```
MOT : constant := SYSTEM.STORAGE_UNIT ;
type APPAREIL is (IMP, LEC, DISQ, DISQFIX, TAMB, BAND, CONS) ;
type ETAT_GENE is (DISPO, OCCUP, INDISPO) ;
type ETAT_PART is array (2 .. 7) of BOOLEAN ;
type NUM_PIS is range 0 .. 8 # 377 # ;
type NUM_CYL is range 0 .. 8 # 377 # ;
type PERIPHERIQUE (UNITE : APPAREIL := DISQ) is
  record
    INTERUP : range 0 .. 8 # 177_777 # ;
    EG : ETAT_GENE ;
    EP : ETAT_PART ;
    MODELE : SORTE_APPAR ; -- نوع غير محدد
  case UNITE is
    when IMP =>
      COMPTEUR_LIG : INTEGER range 1 .. PAGE_SIZE ;
    when DISQFIX | TAMB =>
      POS : NUMPIS ;
    when DISQ =>
      CYL : NUM_CYL ;
      PISTE : NUM_PIS ;
    when others => null ;
  end case ;
end record ;
```



```

for PERIPHERIQUE use
record at mod MOT ;
  EG      at 0 * MOT range 0 .. 1 ;
  EP      at 0 * MOT range 2 .. 7 ;
  UNITE   at 0 * MOT range 8 .. 11 ; -- البتات من 12 إلى 15 غير مستعملة
  POS     at 2 * MOT range 8 .. 15 ; -- المتغيرة DISQFIX
  PISTE   at 2 * MOT range 8 .. 15 ; -- المتغيرة DISQ
  CYL     at 2 * MOT range 0 .. 7 ;
  INTERUP at 1 * MOT range 0 .. 31 ;
-- MODELE,
-- COMPTEUR_LIG حسب خيار COMPILATEUR
end record
end ;

```

تسمح مواصفات تمثيل الفقرات بتكليف برنامج مع سجلات موجودة سابقاً ، أو بغرض تسطير بعض الحقول المستشارة كثيراً كي يتم تخفيض مدة البلوغ بعد السيطرة على الحجم الكبير . هكذا مواصفات يمكن أن تكون كاملة أو جزئية ، وتنطبق أيضاً على فقرات بديلة . في الحالة التي تكون فيها الفقرات عبارة عن جداول ديناميكية ، فإن اللغة لا تعرف شيئاً : يمكن لكل مصنف أن يقدم إمكانية تحليلها .

12.5.2.7 مواصفة العناوين

مثال رقم 20

نفترض إن الرزمة SYSTEM هي مستعملة

```

declare -- تابع المثال رقم 19
LDS1, LDS2, LDS3 : PERIPHERIQUE (IMP) ;
for LDS1 use at SYSTEM.ADDRESS (8 # 777_510 #) ;
for LDS2 use at SYSTEM.ADDRESS (8 # 777_520 #) ;
for LDS3 use at SYSTEM.ADDRESS (8 # 777_530 #) ;
end ;

```

لبرمجة بعض أقسام نواة نظم التشغيل ، فمن الضروري عادة أن يكون هناك متحولات أو متتاليات من التعليمات مزروعة في عناوين مفروضة من قبل العتاد . البرامج التي تستعمل هذه الطرق لتغطية المتحولات (مفهوم التعادل في فورتران) أو متتاليات من التعليمات (مفهوم التغطية ، overlays) هي مغلوبة . هكذا ، فإن الأمان المقدم بواسطة عمليات مراقبة النوع بنهار ، كما يبرهن المثال المحدود التالي :

مثال رقم 21

```

declare
  C : CHAR
  I : INTEGER ;
  for I use at C'ADDRESS ;
  begin
    if I = 8 # 42 # then      -- code (C) × 8 # 42 #
    ..
    end if ;
  end ;

```

.. مغلول : تغطية C بواسطة I

كل من عمليات الترميم هذه التي تستعمل تغطيات بواسطة مواصفة عنوان يمكن أن يتم تفاديه باستعمال وسائل أكيدة مقدّمة من قِبل اللغة أو من قبل بعض المصروفات : نوع مجرّد بتمثيل متعدّد (نوع مشتق) ، رزمة تحويل ، ذرائع خاصة للتغطية .

12.5.3 تقييم

12.5.3.1 النقاط الإيجابية

الإمكانات المقدّمة من آدا لتمثيل المعطيات - التي تشبه في القسم الأكبر منها إمكانيات اللغة LIS : هي جميعها مرئية وواسعة لأنها تسمح بالسيطرة على تمثيل وتوزيع وزرع جميع مواضيع (اعداد ، متحوّلات ، مهام ، برامج ثانوية . . .) اللغة . الوسيلة التي تسمح بإجراء تغيير في التمثيل بواسطة الأنواع المشتقة هي وحيدة . وتساعد دوال عمليات التحويل في زيادة إمكانية العمل وإمكانية الرؤية للبرامج .

12.5.3.2 النقاط القابلة للمناقشة

سنجد ، بالنسبة لمواصفات التمثيل ، بعض التكتّم والاختفاء ، قمنا بصياغته للخصائص والذرائع ، مع أخطار جدية حول فعالية البرامج .

أ - وجود هذه الوسائل في اللغة

من المؤكّد أنها تعقد المصروفات : دور التعريف في اللغة يمكن أن يتحوّل إلى بعض الحبيطة والحلر . وهناك بعض المفاهيم التي لا يمكن أن تكون عامة بالنسبة لجميع المكنات : ترقيم البتات من اليسار إلى اليمين ، إمكانيات التغطية ، الخ . . .

من جهة المستعمل ، فإن استعمال هذه الوسائل له تأثيرات على فعالية التنفيذ . يجب أن نعرف ليس فقط قواعد الاستعمال - وهي معقدة نوعاً ما - ولكن أيضاً طريقة سلوك المصروف المستعمل لتوليد الكود .

في هذه النقطة من التبعة لجهة مفهوم التنفيذ ، كان يمكن أن نؤثر مباشرة على مرحلة توليد الكود دون المرور بواسطة اللغة . (أنظر 12.13 ، 12.2.2.1) .

ب - مبدأ الفصل

مواصفات العنوان هي خطيرة لأنها تسمح بعمليات تغطية وتغليف وإلى تشابه في المواضع . وإستعمال هذه المواصفات ، في الصيغة الجديدة [MRA] ، يوضح بواسطة إدخال النوع SYSTEM.ADRRESS ، وهذا هو تقدم واضح في اللغة . وعلى عكس الصيغة [80 MR] ، فمواصفة التمثيل يمكن أن توضع بالضبط بعد التصريح عن النوع الذي يُحدده . وهذا يسهل عمل المصروفات (بالنسبة للبرامج الثانوية التي تستعمل النوع) ولكن هذا يحدّ عملية الفصل بين الخصائص المجردة والخاصة .

أما تطبيق مبدأ الفصل الجاري في هذه اللغة بالنسبة للأنواع المشتقة فيشكر عليه . فإن مؤلفي آدا بيثون أنفسهم به [ME 13.3.1- M 3.3] .

ونشير هنا إلى أن الفصل هو أكثر فعالية إذا كانت الرزم متماثلة مع أنواع مجردة ، أو بشكل أفضل ، إذا غابت تعابير التكيف عن اللغة .

ج - مشكلة الفعالية

الوسائل المختلفة التي تسمح للمبرمج بتمثيل المعطيات ليست هكذا دون تأثير على فعالية التنفيذ . هذه المشكلة هي حرجة إذا قمنا بتكويد الأنواع المرقمة بواسطة تقوُّب (trons) ، وإذا قمنا بتمثيل الفقرات بواسطة حقول موضوعية في مكان ما في كلمة معينة - أو إذا استعملنا دوال تحويل أو برامج ثانوية مشتقة . مساعد الاستعمال [ME] يهدف للطمأنة حول إمكانية عمل هذه الأولية . ومن غير المؤكد أن يستطيع المصروف اختيار المخطط الأفضل للترجمة إلا إذا كان أمثل وبالتالي هو معقد . وفي كثير من الحالات ، هذه الصعوبات لا يمكن أن تُزال إلا إذا كان المبرمج يفهم بشكل جيد وسيطر على عملية تمثيل المعطيات ووصف المؤثرات التي تعالجها . وهذا هو ممكن ، دون إستعمال مواصفات التكويد من اللغة ، كما يبرهن المثال رقم 22 .

قد يكون من المفائدة للمبرمج أن يوضح بنفسه مخططات الترجمة ، وهذا ما يقدر على إجرائه بسهولة باستعمال الرزم والبرامج الثانوية الأصلية أو «IN LINE» .

مثال رقم 22

declares

```
type UTIL is (CLE_PLATE, CLE_TUBE, CLE_ALIGN, ... SCIE_EGOINE);  
for UTIL use (CLE_PLATE => V1, ..., SCIE_EGOINE := V2);
```

begin

```
for O in CLE_PLATE .. SCIE_EGOINE loop  
  BRICOLER (O);  
end loop;
```

.. تبعاً لتأكيد النوع أداة ، بالإمكان وضع صور ترجمة عديدة ؛

SCHEMA_1 : --V1.. Vj, Vk.. V1 بين اقرب صغيرة وقليلة ..

declare

PRESENT : constant array (V1 .. Vz) of BOOLEAN :=
(V1 .. V1 | Vj .. Vk | ... | Vp .. Vz => TRUE, others => FALSE);

begin

for O in V1 .. Vz loop

if PRESENT (O) then BRICOLER (O); end if;

end loop;

end SCHEMA_1;

SCHEMA_2 : -- كثير من التقريب أو تقريب كبيرة جداً بالمقارنة مع عدد القيم ..

declare

CODES : constant array (1 .. z) of INTEGER := (V1, V2, ..., Vz);

begin

for I in 1 .. z loop

O := CODES (I); BRICOLER (O);

end loop;

end;

SCHEMA_3 : -- وضع ختلط ، حلّ ختلط ..

declare -- لغرض K- عدد التقريب ..

type INTERV is record INF, SUP : INTEGER; end record;

CODES : constant array (1 .. K) of INTERV := ((V1, V1), (Vj, Vj), ..., (Vp, Vp));

begin

for I in 1 .. K loop

for O in CODES (I).INF .. CODES (I).SUP loop

BRICOLER (O);

end loop;

end loop;

end SCHEMA_3;

end;

12.6 مواصفات وصف المعالجات

مراجع : 13.8, 13.9, ME : 14.7

12.6.1 إدخال كود المكنة

12.6.1.1 المبدأ .

من الممكن ، في آدا ، مراقبة توليد الكود بإدخال متتاليات من تعليمات - المكنة في البرنامج . تمثل هذه الأخيرة ببساطة نداءات لإجراءات عمل الخط (البريعة *INLINE*) ، مكدسة بواسطة المبرمج (واحدة أو عدة تعليمات - مكنة لكل إجراء) . النقطة النحوية الجديدة فقط لتنفيذ هذه الأولية - تتمثل في السماح بوجود فقرة تؤمن التأويل اليدوي الذي يرغب به المبرمج . تقوم المكنة المجهزة بعدة أشكال ونسق

للتعليمات ، بإرسال هذه النسق المختلفة من الرزمة SYSTEM . يمكن لهذه الرزمة أيضاً إرسال خاصيات خاصة مُشار إليها في [MRA ملحق F] .

ملحق رقم 23

```

declare
M : MASQUE ;
procedure PLACER ( X : in out MASQUE ) is
  use SYSTEM.INSTRUCTION_370 ; -- رزمة ثانوية من SYSTEM التي ترسل ...
                                -- ترسل عدة نسق تعليمات ، حيث SI-FORMAT
                                -- (Short-instruction-Format)
begin
                                -- أداة لتأويل التعليمة PLACER ، للميزة بواسطة
  SL_FORMAT( CODE => SSM, B => X'BASE, D => X'DISP ) ;
                                -- 370 هي خاصيات مُعلّدة في
end PLACER ;
pragma INLINE(PLACER) ;

begin
  PLACER ( M ) ;
end ;

```

نفس فعالية ماكرو تعليمة .

12.6.1.2 تقييم

فائدة هذه الأوالية المعروضة هي في علم تعميم البرنامج بإدخال كود غريب على اللغة . مبدأ الفصل جرت المحافظة عليه شرط علم إجراء أفعال حرجية كما في الشل [MRA 138] وتفترض هذه الأوالية وجود الخاصيات المُحدّدة لتأويل تعليمات المكتنة .

12.6.2 ملقى مع لغات أخرى

يمكن للبرامج الثانوية المكتوبة بواسطة لغات أخرى أن تُطلب بواسطة برنامج آدا . يجري إعلام المصنف بذلك بواسطة ذريعة معينة ، كي يتم توليد متتالية النداء التي تحافظ على معايير اللغة المقصودة . هذا هو مفيد لترميم البرامج أو مكتبتات البرامج الموجودة ، والملاحظة الثمن لترجمتها إلى آدا . من الممكن أيضاً تنفيذ برامج ترجمة (لغة X ← لغة آدا) ، من اللغات X إلى لغة آدا ، أو بإسكال مثلاً . ولكن في جميع الحالات ، تكون إعادة الكتابة هي الأفضل إذا كانت شروط الكلفة تسمح بذلك .

مثال رقم 24

```

package FORT_LIB is
  function SQRT ( X : FLOAT ) return FLOAT ;

private
  pragma INTERFACE ( FORTRAN, SQRT ) ;

end FORT_LIB ;

```

اللغات المسموحة بواسطة الذريعة INTERFACE تتعلق بالمصرف ويشار إليها في [MRA ملحق F] .

12.7 حلّ عمليات مراقبة المصرف أو المتفدّ

مراجع : ME 14.8 ; MRA 13.10

12.7.1 مبادئ

تقدّم لغة آدا بنائين - يسمحان بوقف أو تحويل بعض عمليات المراقبة للأنواع الساكنة أو الديناميكية : إلغاء التخصيص الجلي والتحويلات الحرة للأنواع . وفي الحالتين ، يجب إجراء توليد لرزمة أصلية خارجة من مكتبة محيط البرمجة (رزمة غير نموذجية NO STANDARD) .

12.7.1.1 إلغاء التخصيص الواضح

نستعمل الرزمة UNCHECKED-DEALLOCATION . ولقد أظهر المثال رقم 14 من الفقرة 7.4.1 الإستعمال الطبيعي لهذه الرزمة . المثال المحدّد التالي يظهر الأخطاء الناتجة عن سوء الاستعمال .

مثال رقم 25

- نرى الرزمة UNCHECKED-DEALLOCATION

declare

```
type NOEUD is ... ;
type ARBRE is access NOEUD ;
procedure RENDRE_NOEUD is
  new UNCHECKED_DEALLOCATION ( NOEUD, ARBRE ) ;
A1, A2, A3 : ARBRE ; A : constant ARBRE := new NOEUD ;
```

begin

```
A1 := new NOEUD ; A2 := A ; A3 := A1 ;
```

- لنفترض إنّ N1 و N2 هي مواضيع يُشار إليها بواسطة A1, A2, A3 و A.

- التعليمات التالية هي غير مسموحة أو مغلوطة

```
RENDRE_NOEUD ( A ) ;
```

- غير مسموح ، لأن A هي ثابتة

```
RENDRE_NOEUD ( A2 ) ;
```

- N2 هو رسمياً حرّ بينما يتم مراجعته بواسطة A .
- يمكن أن يُسلم بواسطة تخصيص آخر .

```
RENDRE_NOEUD ( A1 ) ;
```

- N1 رسمياً حرّ ، يمكن أن يُراجع بواسطة A3

```
A1 := new NOEUD ;
```

- A1 يعاين موضوعاً جديداً ، محتملاً N1

- (مراجع بواسطة A3) أو N2 , (مراجع بواسطة A) .

end ;

12.7.1.2 تحويلات الأنواع غير المفحوصة

بدلاً من إستعمال عمليات تغطية التحولات لإنشاء تحويلات غير مفحوصة ،
سنستعمل توليد للإجراء UNCHECKED-CONVERSION .
مثال رقم 26

.. نرى الرزمة UNCHECKED-CONVERSION

```
declare '  
  C : CHAR ;  
  N : INTEGER ;  
function CHAR_INT is  
  new UNCHECKED_CONVERSION ( CHAR, INTEGER ) ;  
begin  
  N := CHAR_INT ( C ) ; - = de CHAR'POS ( C ) ou de CHAR'IMAGE ( C ) .  
end ;
```

12.7.2 تقييم

هاتان الإمكانتان ، اللتان كانتا تدعيان في [GREEN] « ضعيفتي الإمكانية في العمل » ، كانتا تسميان قبل ذلك « غير قابلتين للتحكم » . وهذا يتناسب مع الحقيقة : فمن الممكن كتابة برامج صحيحة تستعملها . في أغلب الأحيان ، كنا نلاحظ من خلال هذين المثالين السابقين جميع الأخطار الملازمة لاستعمالها . في هذه المرة ، يدلونا إيجابياً أن نكون ملزمين بتثبيت وحدات التصريف التي تستعمل الرزم « الخطيرة » ، مع الجملة with . وهذا يسمح بلفت انتباه المسؤول عن المشروع ومنع إستعمال هذه التسهيلات .

12.8 خاتمة

لتكثيف برنامج مجرد مع مختلف شروط محيط التنفيذ ، تتبع آدا مفهوماً كلاسيكياً بتقديم ، في نحوها ، بيانات تسمح بحوار ثابت بين المبرمج والمصرف من أجل مرحلة توليد الكود . الامتحان التفصيلي لهذه الإنشاءات أثبت أنها مفيدة ، غنية في الإمكانيات ومقروعة . وتسمح باستعمال آدا لكتابة أي قسم من النظام ، لأي مكنة مهما تكن ، مع الإستفادة من إمكانيات لغة عالية المستوى ، منوعة : لقد إستفادت آدا من الأبحاث التي جرت خلال السنوات العشر الماضية على لغات كتابة الأنظمة ، وبشكل خاص على اللغة LIS .

وسائل تكثيف لغة آدا تؤدي إلى بعض المفاهيم القابلة للجدال والتي سنوجز النقاط الأساسية منها .

إنشاءات ذات دور مهم

تختلف مهمات الإتصال بين المبرمج ومحيطات البرمجة والتنفيذ (أنظر 12.1.4 ،

شكل 1) ليست مُثَلَّة بواسطة إنشاءات بدور مُعَدَّ جيداً . تستخدم الخاصيات لإعلام المبرمج ، ولتعريف حالة المهمة . وعلى العكس ، يوجد عدة وسائل للتأثير على الترجمة (ذرائع أو مواصفات تمثيل) أو لرفع بعض عمليات التحكم (ذرائع أو إستعمال الرزم) . هذا ، فالترقيق بين الذريعة ومواصفة التمثيل يمكن أن يتم بواسطة قاعدة بسيطة : مواصفة إلزامية أو إختيارية .

مبدأ الفصل بين الصفات المجردة والخاصة

في 12.1.2 قمنا بالتذكير بأهمية حصر تمرکز مميزات التكييف . وفي آدا ، يمكن أن تكون الخاصيات مستعملة في أي قسم من البرنامج ، حتى في تلك التي تُعرَف على المميزات الفيزيائية (عنوان تسجيل متحولة مثلاً) . أما الذرائع فلا يمكن أن توضع في أي مكان ، واستعمالها يخضع لقواعد معقّدة تتعلّق بالذريعة . أما بالنسبة لمواصفات التمثيل ، فيمكن أن تظهر بالبسيط بعد التصريح عن الأنواع التي تحددها . يمكن للمبرمج أن يقوم بتنظيم مكان التسجيل الفيزيائي لتحولاته ، وأن يقوم بالتغطية ، وكل هذا في أقسام داخلية .

بينما ، في الصيغة الأخيرة للغة [MRA] ، فإن إستعمال أوليات شديدة الخطورة هي دائماً جلية في رؤوس وحدات التصريف التي تستعملها ، بواسطة with (SYSTEM ، UNCHECKED-DEALLOCATION ، UNCHECKED-CONVERSION) ، بواسطة إستعمال الخاصية ADDRESS أو مواصفة عنوان . وهذا هو تقدم واضح ، لأن هذا يسمح بالتحكم باستعمال هذه الإنشاءات . على العكس ، فإن إمكانية وضع مواصفات التمثيل بشكل حرّ أكثر يبدو لنا وكأنه تراجع ، لأن هذا يضعف الفصل بين الخصائص المجردة والخاصة للنوع . وفي أغلب الأحيان ، تكون جميع هذه الوسائل قابلة للتصريف بواسطة النحو ومن الممكن إنشاء أدوات البرمجة ، توضع وتتحكم باستعمالها .

لغة مركّزة

كأغلب اللغات الحالية ، تقدم آدا تشكيلاً موحداً لمختلف النشاطات في البرمجة : برمجة مجردة ، تثبيت متغيرات ، شمولية ، تكييف ، كتابة ، قراءة وتحكم بنص البرنامج الخ . ولقد أشرنا في 12.1.3 للفوائد والسيئات الملازمة لهذه الصيغة المركّزة . وفي حالة آدا ، فإن النتائج على التعقيدات هي مؤسفة لأن هذه اللغة لا تحتوي على صفة الإسقاطية إلا قليلاً [BOUT 80] . وتعلمها هو صعب ويخشى من إن قلة من الأشخاص سيفهمونها بشكل كامل .

ومن جهة أخرى ، حتى مع كون آدا لا تقدم أية سهولة جديّة للتوثيق والعرض ، فلم

يكن من الضروري تقديم وسائل تكييف في اللغة ، وحتى نفس الإمكانات للشمولية . وبدلاً عن ذلك ، فالتنظيم في عدة مستويات غرضية كان سيسمح بتقديم خدمات عديدة - بما فيه ما هو مفقود حالياً - بدون أن يحدث ذلك تعقيداً زائداً مفروضاً على جميع المصنفات وعلى أغلب المبرمجين . وإذا كان حقاً إن جميع الأشخاص الذين يكتبون برامج بلغة آدا ليس بإمكانهم معرفة اللغة بالكامل ، فهؤلاء الذين يقرؤنها - وبشكل خاص أولئك الذين يقومون بصيانتها - يجب أن يعرفوها بالكامل . فمع تنظيم في مستويات مختلفة ، مستطيع وسائل التكيف أن تقدّم مستوى خاصاً ، لا يعطى أوامر إلا في مرحلة التكيف (توليد الكود الأفضل) ، مما يخفف من مراحل البرمجة المجردة ويسهل من إمكانية النقل من مكانة إلى أخرى .

دور اللغة

جرى تصوّر آدا بالطرق والوسائل الخاصة بأكبر الصناعات الإنسانية [Rault 79 , Wegner 80] . ولكن ، يبدو أننا وضعنا العربية بعد الحصان بتعريف اللغة قبل تعريف الوسائل التي تستعملها . وهذا يشرح ، برأينا ، بعض الضعف في آدا : دورها المركز والعمومية الموجودة حول موضوع العلاقات بين اللغة ووسائل البرمجة . هكذا ، فاللغة تمجد في أن تكون مستقلة عن هذه الوسائل ، ولكن هذا لا يتم بشكل حقيقي : هناك عدة مفاهيم للغة تقوم بمراجعة هذه الوسائل (ترتيب الأخطاء ، التصريف المنفصل ، الشمولية ، وسائل التكيف) .

على العكس ، فنحن نفكر إنه كان من الأفضل تعريف مهام الوسائل الضرورية للبرمجة أولاً (في المعنى العريض) ، من خلال تحليل شامل لمختلف النشاطات ، واللغة كانت ستتقدّم بواسطة نظام متماسك من اللغات لتحمّل مختلف الاتصالات بين هذه الوسائل والمبرمجين . هكذا فحقاً إن مفهوم كهذا ، معروف لتصور أنظمة المعلومات ، ما يزال قليل الاستعمال لتصور الوسائل في علم المناهج ، ويقع على الحدّ الفاصل بين حالة الفن الحالي .

بالرغم من هذه السيئات ، فإذا قمنا بمقارنة آدا مع اللغات الأخرى الموجودة اليوم ، فيمكننا بدون تردد أن نضعها أفضل اللغات لكتابة الأنظمة . السيئات السابقة تبرهن إن البحث على اللغات يبقى مجالاً مهماً ويوماً .

الفصل الثالث عشر

المدخل - المخارج

13.1 المدخل

13.1.1 ما يتكوّن هذا الفصل ؟

شكلت عمليات الإدخال - الإخراج المفهوم الأكثر تغيّراً في لغة آدا من صيغة إلى أخرى . وبينما عرفت اللغة مجموعة بعض الثبات ، الذي تأكد بين صيغة ثموز 80 [MR] والصيغة 82 [MRA] ، وعلى العكس ، فمن الممكن إعتبار وجود ثلاثة تعريفات متتالية في لغة آدا ، كل تعريف منها له مميزاته ، فوائده وسيئاته . وإذا كانت بعض عمليات الاختيار الأساسية في اللغة لم تتغيّر ، وبشكل خاص تلك التي تقوم على عدم إضافة إمكانيات جديدة الى اللغة لادخال عمليات إدخال - إخراج جديدة إليها (أنظر 13.1.3 و 13.3.4) ، وإذا كانت بعض الصيغ قد صممت ففي المقابل ، إن مجموعة الإمكانيات المقدمة من اللغة قد أصابها تعديلات مهمة بين الصيغة 79 [GREEN] وصيغة ثموز 80 [MR] .

الحالة الأولى لهذا الفصل ، المتّفق إعتباراً من الصيغة 80 ، كانت الأكثر حرجاً في هذا الكتاب . فعرض الإدخال - الإخراج المعمول به في لغة آدا كان يبدو لنا وكأنه لم يؤخذ به بالكامل ، وقمنا بإجراء بعض التعريفات الجديدة لأسس جديدة للإنطلاق . أما منطلقنا فلقد أصابت فريق العمل في تطوير هذه اللغة ، وجان إيشابيه Jean Ichahiah اقترح لأحدنا (OL) بالحضور للمشاركة في فترة العمل على إعادة تعريف بعض النقاط الأكثر إنتقاداً . حالياً فإن حالة الادخال - الإخراج قد أخذت بالحسبان العديد من إنتقاداتنا التي قمنا بها . وفي بعض النقاط الأخرى فإن آراء مؤلفي اللغة بقيت مختلفة عن آراء المُقيمين وهذا ما يلاحظ في باقي هذا الفصل .

لتقديم وسائل الادخال - الإخراج لمستعملي اللغة آدا ، فإن مؤلفيها وضعوا إختيارين أساسيين يتعلّقان بما يجب توضيحه ، وبطريقة هذا التوضيح . الإختبار الأول هو نموذج وصفي وبجود لمفهوم الإدخال - الإخراج ، والثاني يقوم على تعريف الإدخال -

الإخراج بواسطة اللغة نفسها ، دون إضافة أي مفهوم ، بواسطة مجموعة رزم محددة سابقاً . هذان الإختياران هما مستقلان الواحد عن الآخر ، وسيتم توضيحهما وتقييمهما بشكل منفصل .

13.1.2 نموذج مفهوم الإدخال - الإخراج

يمكن تمييز ثلاثة مستويات للإدخال - الإخراج ، نسميها مستوى المحيطات (مداخل - مخرج بمستوى منخفض ، في [MR] ، مستوى نصوص (مداخل - مخرج نصوص) ومستوى سجلات (مداخل - مخرج عامة في مستوى المستعمل) . مجموعة النقاشات اللاحقة ستقوم بفصل هذه المستويات الثلاثة .

13.1.2.1 مستوى السجلات [MR 14.1 ؛ MR 15.2]

هذا المستوى يتركز على مفهوم السجل ، بتمييز السجل الداخلي ، الذي يصرح عنه ويعالجه البرمج ، والسجل الخارجي ، الذي يرتبط به السجل الأول . هذا الربط هو متعلق بالعمل ، ويتم في لحظة فتح السجل ، بسبب المعلومات المقلّمة بواسطة سلسلتين من السمات : اسم وشكل السجل الخارجي . طول ومضمون هاتين السلسلتين يتعلّق بعمل المكنة .

السجل الداخلي هو موضوع يجب أن نُصرّح عنه من البرنامج ، وهذا الموضوع أو الغرض ذو نوع معين يدخل في هذا النوع نوع المركبات وطريقة التنظيم ، المتتالية أو المباشرة . ولا يدخل فيه اتجاه الإرسال ، التي تتمتع بخاصية الموضوع والتي يمكن أن تُقدّر ديناميكياً .

من الممكن القول تقريباً إن السجل الداخلي هو نموذج لتصوير « جدول وصف السجل » ، الذي نجده في أغلب أنظمة الإدخال - الإخراج .

السجل الخارجي ، الذي يؤلف السجل الداخلي ، مجرد وسيلة لبلوغه ، هو عبارة عن الناقل الفيزيائي لمركبات السجل ، لمدة حياة مستقلة عن حياة البرنامج . ونوع السجل الداخلي المختار هو الذي يحدد كيفية اعتبار السجل الخارجي ومعالجته . يمتاز السجل المتتالي بخصائص سلسلة عادية ، والسجل المباشر بخصائص جدول يبدد واحد وبدلائل صحيحة وإيجابية . ومع إن هذه السجلات هي بسيطة وعامة ، فإن مجموعة وسائل الاتصال بين السجل الداخلي والخارجي تدل على إنها معتمدة لنقل محطتي وليس لوصف مجموعة من المواضيع الداخلية في البرنامج .

13.1.2.2 مستوى النصوص [MR 14.2 ؛ MR 15.3]

هذا المستوى هو ضروري للمداخل - المخارج المرئية بواسطة الكائن البشري وللاتصالات . في هاتين الحالتين ، فإن استعمال السجل المتتالي حيث المركبات هي عبارة

عن سمات ، سيكون صعباً . النموذج المختار يعالج سجلاً من النصوص كلائحة من السمات ، التي يمكن أن تكون مجموعة في أسطر ، أما سلاسل الأسطر فيمكن أن تجمع في صفحات . الاستعمال هو متتالي محض ، واتجاه الارسال يُعالج كما بالنسبة إلى مستوى السجلات .

مفهوم النسق ولائحة الإرسال ، المخترعة في لغة فورتران ، والمعتمدة في Algol 68 ، والداخلية كجملة أصلية *intrinsic* في مفهوم الإدخال-الإخراج ، ليست مستعملة ، لحساب طريقة عمل مشابه لعمل لغة باسكال : تنقل المعلومات بشكل مُستقل ، وطريقة استعمالها وتكوينها مُحدّد بواسطة النوع وبواسطة بعض المؤشرات الثانوية .

وفي الإمكان تنظيم البنية بالأسطر وبالصفحات بشكل واضح بسبب الأوامر المخصصة لذلك . كما ويمكن أن تهمل في الإدخال ، وتنظّم عند الإخراج وذلك بواسطة التعريف الذي يقوم به المبرمج للأطوال المحدّدة .

13.1.2.3 مستوى المحيطات

هذا المستوى يتعلّق بشكل كبير بالعمل ، والمؤلفون لم يُعرّفوا سوى الخطوات الواجب إتباعها لتأمين وسائل بلوغ إلى المحيطات . لا يوجد لا مفهوم للسجل ولا مفهوم للفتل أو لوحدة التبادل . - فقط الأوامر *primitives* هي المستعملة لتأمين التبادل بالمعلومات بين وحدة التحكم والمحيط ويشكل غير محدّد مسبقاً .

13.1.3 لا يوجد إضافات إلى اللغة [MR 14, ME 15.1, 15.5]

مفهوم الإدخال-والإخراج هو معقّد ، ولا يوجد عتلاً أية لغة للبرمجة يمكن أن تُؤكّد بأنها بسيطة وطبيعية بشكل كامل . بالنسبة للغة آدا ، فإن تعريف أوامر الإدخال والإخراج لا ييلو وكانها تحتاج إلى إضافات .

حسب [ME 15.1] ، فإن هذا الاختيار قد تم لتضادي صعوبة هذه اللغة وزيادة حجم الأعمال بإضافة إمكانيات متعددة ومختلفة ، وحسب تعدّد وتعقيد مفهوم الإدخال-الإخراج . السبب الآخر المتقدم في [ME 15.1] هو في إن هذا الاختيار يسمح لمستعمل خاص ، في حال كانت مجموعة الأوامر المحدّدة لا تكفي ، بتعريف واحد يناسبه وجود مجموعة محددة هو إثبات أن ذلك هو ممكن .

من الممكن تصوّر سبب آخر . تُشكل عمليات الإدخال-الإخراج إحدى النقاط الأكثر تأثيراً بالتطور التكنولوجي ، ومن الممكن أن نتصور أنه ، في لغة معتمدة للاستعمال الطويل ، بإمكاننا ترك إمكانية تكيف بعض النقاط حسب هذا التطور ، وذلك بتعريف الرزم النموذجية في كل مرة نشعر بالحاجة إلى ذلك . هذا لن يمنع أبداً ، إضافة إمكانيات جديدة لتسهيل برمجة هذه الرزم إلى اللغة . فلنشير إلى إن عمليات الإدخال-الإخراج هي

مقدمة كقسم متكامل مع اللغة لأنها موصوفة في فصول خاصة بها [MR] و [ME] ، وليست في ملاحق .

من الممكن إضافة سبب جديد وهو ؛ في ان هناك نزعة لاستعمال لغة آدا في كتابة الأنظمة ، وبشكل خاص في كتابة أنظمة التشغيل أو أنظمة إدارة السجلات ، إذا كانت مزودة ببيانات إدخال - إخراج ، وفي جميع الأشكال من الواجب صنع نظام إدارة السجلات متداخل في نظام ثانوي للغة ، يفترق إلى الإدخال - الإخراج . هذا الحل يُسهّل إنشاء العمليات .

13.2 الشروحات

سنحاول في هذا الفصل إيجاز كل ما تقدّم ، بمصاحبة أمثلة مفصلة . المساعد المرجعي يقدم مواصفة غير إلزامية لرزم الإدخال - الإخراج ، وذلك بوصف الوجهة النحوية بشكل دقيق ، في لغة آدا ، والوجهة الدلالية بالإنكليزية .

13.2 مستوى السجلات

السجل هو مفهوم مجرد يتمثل في وحدتين : السجل الداخلي (مجرد) الذي يؤلف قسماً من البرنامج ، والسجل الخارجي (خاص) الذي يخرّن قيم السجل المجرد . هذه الوحدات هي مرتبة في البرنامج ، الذي يستطيع ربطها ، ومعرفة حالتها ومعالجتها .

السجل هو مجموعة من العناصر من نفس النوع ، بلوغ هذه العناصر يمكن أن يتم بشكل متتالي ، حسب نموذج السلسلة (سجل متتالي) ، أو بشكل متتالي أو إنتقائي . ، حسب نموذج الجدول (سجلات مباشرة) . معالجة السجلات تتم بواسطة أوامر منقولة بمساعدة نماذج لرزم أصلية SEQUENTIAL-IO (سجلات متتالية) أو DIRECT-IO (سجلات مباشرة) . يلزمنا عدد من هذه الأمثلة على عدد الأنواع المختلفة لعناصر السجلات .

يُعرف السجل الخارجي بواسطة سلسلتين من السمات ، إسمه وشكله ، حيث يتعلّق النحو بنظام التشغيل المستهدف . الإسم يمكن ، مثلاً ، أن يكون مؤلفاً من عدة حقول ، كما هو الحال عادة بالنسبة لأنظمة سجلات الميكروحاسبات (في RT11 ، مثلاً ، إسم السجل هو بالشكل (PERIPHERIQUE : NOM. SUFFIXE) ، أو في أنظمة السجلات التراتبية (في النظام UNIX ، إسم السجل يعني إسم المسار بين الجذع والسجل ، مثلاً : PROJ / PROGRAMEUR / FICHIER / SUFFIXE ; / USER / LABO .

النظام Multics ومشتقاته يستعمل فكرة شبيهة بذلك . كما ويمكن أن يعني الشكل نوع السجل ، بميزات ناقل السجل ، وحقوق البلوغ المسموحة ، الخ . يمكن للسجل

الخارجي أن يكون قابلاً للمعالجة مباشرة بواسطة أوامر المعارف التابعة لنظام التشغيل أو لنظام السجلات الموجود .

لا يمكن لبرنامج بلغة آدا أن يكون له حق البلوغ إلى عناصر السجل إلا بواسطة أوامر تنطبق على السجل الداخلي ، وفقط عندما يربط هذا الأخير بالسجل الخارجي . من الممكن تمييز ثلاث فئات من الأوامر التي تسمح بإجراء هذا الربط ، ومعرفة حالة السجل أو المربوط ، بإجراء البلوغ إلى العناصر . جميع هذه الأوامر تطبق على السجلات المتتالية والمباشرة ، ما عدا عمليات البلوغ المباشرة الموجودة فقط للسجلات من نفس الاسم .

أ - أوامر إدارة الربط بين سجل داخلي وسجل خارجي
CREATE - يقوم بإجراء ربط بين السجل الجديد ويخصص المكان للسجل الخارجي .
OPEN - يقوم بإجراء ربط للسجل الخارجي المنشأ سابقاً .
CREATE و OPEN يمكن أن تدل على طريقة الاستعمال ، التي قد تكون IN-FILE (قراءة فقط) ، OUT-FILE (كتابة فقط) أو INOUT-FILE (قراءة وكتابة) ، أي إستيفاء يومي) . بعد الفتح ، نفس السجل يمكن أن يقسم على التوالي بين عدة عمليات (أي ربطه بعدة سجلات داخلية) ، إذا كانت حقوق البلوغ المشار إليها بواسطة شكل وطريقة الاستعمال - هي متكيفة مع حالة السجل الخارجي ؛ في الحالة العاكسة ، فالحالة - الإستثنائية USE-ERROR سيتم إطلاقها .
RBSSET - تسمح بإعادة تصفير وإعداد حالة السجل ، ومختلاً بتغيير طريقة استعماله ، دون قطع ما هو مربوط سابقاً .
CLOSE تنهي الربط الجاري وتغلق السجل .
DELETE ينهي الربط الجاري ، ويجرؤ المكان المشغول بالسجل الخارجي الذي لا يبقى له أي وجود .

ب - أوامر فحص حالة السجل
من الطبيعي أن يعرف البرنامج حالة سجلاته . ويمكن أن يقوم السجل بلغو المتغير لدى إحدى الإجراءات ، وقد يحدث أن يحمل هذا الأخير حالة السجل المنقول ، ويتمنى معرفته بشكل آخر دون ترميم الحالات الإستثنائية التي تشير إلى هذا التناقض . الأوامر المتتالية تسمح بمعرفة حالة السجل الداخلي وخصائصات السجل الخارجي المتعلق به .

MODE تعيد طريقة الفتح الجارية .
NAME تعيد إسم السجل الخارجي المربوط .
FORM تعيد شكل السجل الخارجي المربوط .
IS-OPEN تدل ما إذا كانت الوصلة قد تمّت .

END-OF-FILE تدل على إن القراءة المتتالية الحالية قد انتهت وعلى إن أمراً جديداً بالقراءة المتتالية هو ممنوع .

SIZE تدل على عدد العناصر المخصصة للسجل الخارجي ، في لحظة النداء .

INDEX تدل على قيمة الدليل الجاري .

هذه الأوامر الأخيرة هي غير موجودة إلا في السجلات المباشرة .

ج - أوامر البلوغ المتتالية للعناصر

READ يقرأ العنصر التالي إذ لم يتم الوصول إلى نهاية السجل : not-END-OF-FILE .

WRITE يكتب عنصراً جديداً في نهاية السجل .

هـ - أوامر البلوغ المباشرة للعناصر

السجل المباشر هو شكل من أشكال الجدول الموسع بالعناصر ، حيث كل عنصر يُشار إليه بواسطة دليل رقمي صحيح وإيجابي . الدليل الجاري ، الذي يعد عند الفتح وذلك بحصوله على القيمة 1 ، وتزداد قيمته بعد كل عملية قراءة أو كتابة ، والتي تبدأ بواسطة الأمر SETINDEX ، يدل على العنصر الذي نبلغه بواسطة عملية قراءة أو كتابة بدون قيم واضحة للدليل .

يمكن أن تكون عمليات القراءة والكتابة مباشرة (بذكر الدليل بشكل واضح) ، وإما بالتالي (باستعمال الدليل الجاري الاستعمال) .

الجدول 1 يوجز تأثيرات عمليات البلوغ الممكنة إلى سجل مباشر .

ملاحظات :

- من الممكن قراءة عنصر غير محدد ، ويمكن لها أن تطلق DATA-ERROR ، حسب الأعمال .

- الاستيفاء البرمي لـ (%) بواسطة CLOSE ، OPEN ، WRITE و SET-INDEX ليس محدداً في الوثائق التقريبية التي قمنا باستعمالها . من الممكن أن نفهم إنه غير ممكن ، أو أن مفعوله يتحدد بالاستعمال .

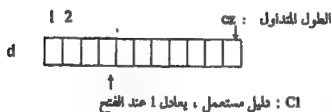
13.2.2 مثال رقم 1 تقييم السجلات المتتالية المنظمة .

لقد قمنا باختيار هذا المثال الكلاسيكي لتسهيل المقارنة مع اللغات الأخرى . لقد أردنا تطوير حل واقعي يتطابق مع ذهنية اللغة ، باستعمال جميع إمكانياتها . وهذا ما يؤدي ، في هذا المثال كما في الأمثلة السابقة ، إلى توضيح استعمال الشمولية ومعالجات الحالات الاستثنائية .

نقطة الإنطلاق هي في كتابة - إجراء ضم بين سجلين متتاليين منظمين F1 و F2 ،

يجعل النتيجة في F3 . استعمال الرزمة الشاملة SEQUENTIAL-IO ، المثبتة بواسطة نوع العناصر ، يفترض إتباع نفس العملية مع إجراء الضم . في هذه الحالة ، فإن مؤثر مقارنة العناصر يُقدّم كمتغيّر للشمولية ، وهذا يعطي إمكانية استعمال نفس نموذج الضم للرتب المتصاعدة والمتناقصة ، وهذا ما يسمح أيضاً بالتجرّد الكامل . عن تركيبة العنصر : قد يكون سهلاً أو مركباً ، لا إنتهاهي محمّداً أو غير محدد ، أن يتمتع بواحدة أو عدة مفاتيح ، الخ .

سجل مباشر جدول قابل للتوسيع بالعناصر



d : سجل مباشر من العناصر .

e : عنصر من السجل .

i : مؤشر أو دليل .

	قراءة	كتابة
بلوغ متتال	<pre> READ (d,e) اللفعل: si ci > cz alors END_ERROR sinon e := d (ei) ; ci := ci + 1 ; fin si </pre>	<pre> WRITE (d,e) اللفعل: d (ei) := e ; ci := ci + 1 ; </pre>
بلوغ مباشر	<pre> READ (d,e,i) اللفعل: si i > cz alors END_ERROR sinon e := d (i) ; ci := i + 1 ; fin si </pre>	<pre> WRITE (d,e,i) اللفعل: d (i) := e ; ci := i + 1 ; </pre>
الدليل المتداول	<pre> return ci (d) دليل </pre>	<pre> SET_INDEX (d,i) اللفعل: ci := i ; </pre>

جنول 1 عمليات البلوغ الممكنة لسجل مباشر

تفرض اللغة أيضاً ذكر الانتهاء الخارجي لأوامر معالجة السجل التالي : READ ،
WRITE ، END-OF-FILE . نلاحظ إذا إن إجراء الضم يمكن أن يعمل :

- مع السجلات المباشرة ، المجهزة أيضاً بنفس مجموعة الأوامر (أنظر 13.2.2.4) .
- بشكل عام ، مع كل بنية مجردة ترسل مجموعة معادلة لأوامر المعالجة المتتالية (أنظر
13.2.2.5) .

كي يكون صالحاً للإستعمال بواسطة أجزاء الضم ، فإعداد البنى المجردة يمكن أن
تتم بواسطة محيط الإجراء . هكذا ، فعمليات الأعداد والتصغير هي متعلقة بنوع البنية .
إضافة لذلك ، فمعالجات الحالات الإستثنائية لا تتم في إجراء الضم ، الذي لا يقوم إلا
بنشرها .

شروط وقف القراءة يمكن أن تتناسب مع تدمير البنية المقروءة ، وهذا يسمح
باستعمال إجراء ضم في خوارزم للفرز بواسطة ضم الأحاديث ، مثلاً .

إجراء الضم الذي يتبع هو عام ، والقارئ سيتحقق من إنه ، في أغلب الحالات ،
لن يؤدي هذا إلى خسارة في فعالية التنفيذ ، بالنسبة للإجراء على القياس . وبإستطاعتنا
جعله أكثر عمومية ، وذلك بالتعريف في متغيرات الشمولية ، ليس فقط عن البنية المجردة
المشتركة بالسجلات الثلاثة F3, F2, F1 ، ولكن عن ثلاث بنى مختلفة . هذا سيسمح ،
مثلاً ، بضم مكدس وسجل متتالي في جدول . لم نقم بذلك كي لا نعقد المثال أكثر من
ذلك .

13.2.2.1 - مواصفة إجراء الضم

ضم بنيتين S1 وS2 (أنواع مجردة) ، مقروءة على التوالي ، ومنظمة حسب علاقة
« \ll » . والنتيجة هي البنية S3 التي نستطيع تعبئتها على التوالي ؛ إعادة القراءة المتتالية لـ S3
تؤدي إلى إخراج جميع العناصر من S1 وS2 ، منظمة حسب العلاقة « \ll » .

تستخدم لقراءة S1 وS2 : LIRE et FIN-DE-LECTURE .

تستخدم لتعبئة S3 ، الذي يمكن ECRIRE ، الذي يمكن
أن يكون فارغاً .

من المفترض أن تكون البنى S3, S2, S1 معدة ومهيبة بشكل صحيح للمؤثرات
السابقة في الصيغة الحالية ، يجب أن تكون S3, S2, S1 من نفس النوع الشكلي
STRUCT ، والأجراء FUSIONNER لم يبين بشكل متين . وهي تؤمن إستعمال صحيح
للمؤثرات المنقولة ، ولكنها لا تعالج أبداً الحالات الإستثنائية التي يمكن أن تطلق بواسطة
المحيط عند إستعمال هذه المؤثرات ؛ هذه الحالات الشاذة تنتشر ويجب أن تعالج في المحيط

الذي يستعمل FUSIONNER . السبب وإسم كل من هذه الحالات الإستثنائية يجري شرحه في مواصفات البنى المستعملة .

generic

type ELEM is private ; - عناصر البنيات S1 ، S2 ، S3 -

with function " < " (X, Y : in ELEM) return BOOLEAN is <> ;

- مؤثر حاسم للعلاقة « > » -

type STRUCT is limited private ; - نوع مجرد لـ S1 ، S2 ، S3 -

with procedure LIRE (S : in STRUCT ; E : out ELEM) is READ ;

- سابق : not FIN_DE_LECTURE (S)

- ناتج : E = عنصر S ، « العناصر التي يحتمل أن تكون قُرئت على S »

with function FIN_DE_LECTURE (S : in STRUCT) return BOOLEAN
is END_OF_FILE ;

- « بعد هذه S عناصر » العناصر التي يحتمل أن تكون قد قُرئت انتهاء البنية أو انتهاء التسلسل .

with procedure ECRIRE (S : in STRUCT ; E : in ELEM) is WRITE ;

- سابق: S هو جازم لاستقبال عنصر جديد ELEM

- ناتج S: استقبال عنصر جديد « من كل أولئك الذين قد إستقبلهم منذ أن أصبح الإجراء FUSIONNER عاملاً وفعالاً .

procedure FUSIONNER (S1, S2, S3 : in STRUCT) ;

ملاحظة

الأوامر LIRE و ECRIRE ، التي تغيّر الحالة المجردة لـ S ، تنقل S في الصيغة in . وهذا ضروري كي نستطيع أن نضيف إليها الأوامر READ و WRITE ، المنقولة بواسطة الرزم SEQUENTIAL-IO و DIRECT-IO . إجراء الضم السابق لا يمكن أن يُستعمل كما هو (يجب إضافة إجراءات أخرى إلزامية كتحميل زائد) للمكندس المحنّد في :

procedure PUSH (S : in out STACK ; E : in ELEM) ; - WRITE

procedure POP (S : in out STACK ; E : out ELEM) ; - READ

نعتقد إن هذا هو خطأ بتحديد الصيغة in في إجراءات معالجة السجلات التي تغيّر حالة السجل (READ ، WRITE ، SET-INDEX ، الخ) حتى إذا كانت أنواع السجلات (FILE-TYPE) تناسب بالفعل أداة وصف معينة . نفس الشيء بالنسبة لمؤشر يدل على بنية معينة ، إذا كان هذا المؤشر يمثل للمستعمل البنية نفسها .

الصيغة IN OUT ستظهر لنا مفصلة في كل وجهة نظر :

- فهي تتطابق مع الفكرة المجردة التي يقوم بها المستعمل .
 - تسمح باستعمال رزم الإدخال - الإخراج مع تركيبات أخرى للمعطيات ، في إجراءات أو رزم عامة (ضم ، فرز ، إستيفاء يومي ، الخ) .
 - لا تقدم أي خطر في معالجة مخلوطة إذا كان النوع الذي يعني التركيبة هو محدود خاص (وهذا حتماً هو النوع FILE-TYPE) .
- 13.2.2.2 دوران إجراء الضم

```

procedure FUSIONNER (S1, S2, S3 : in STRUCT) is
    T1, T2, T3 : ELEM ; -- S3 و S2 ، S1 أمثلة مخصصة لـ
    T1_DEFINI, T2_DEFINI : BOOLEAN ;

    procedure COPIER_RESTE (S : in STRUCT ; PREM : in ELEM) is
        "S3 ← S3 & PREM & RESTE (S)"
        E : ELEM ;
        begin -- يفترض PREM معرفة :
            ECRIRE (S3, PREM) ;
            while not FIN_DE_LECTURE (S) loop
                LIRE (S, E) ; ECRIRE (S3, E) ;
            end loop ;
        end COPIER_RESTE ;

begin -- FUSIONNER
    if not FIN_DE_LECTURE (S1) then
        LIRE (S1, T1) ; T1_DEFINI := TRUE ;
    else T1_DEFINI := FALSE ;
    end if ;
    if not FIN_DE_LECTURE (S2) then
        LIRE (S2, T2) ; T2_DEFINI := TRUE ;
    else T2_DEFINI := FALSE ;
    end if ;
    while T1_DEFINI and T2_DEFINI loop
        -- أقرب عنصر = (min (T1, T2)) : لا يتغير :
        -- "مقدمة عند ابتكار FUSIONNER" :
        if T1 < T2 then
            T3 := T1 ;
            if not FIN_DE_LECTURE (S1) then
                LIRE (S1, T1) ;
            else T1_DEFINI := FALSE ;
            end if ;
        else
            T3 := T2 ;
            if not FIN_DE_LECTURE (S2) then
                LIRE (S2, T2) ;
            else T2_DEFINI := FALSE ;
            end if ;
        end if ;
        ECRIRE (S3, T3) ;
    end while ;

```

```

end loop ; -- affirme not T1_DEFINI xor not T2_DEFINI
-- يؤكد not T1_DEFINI or not T2_DEFINI
if T1_DEFINI then COPIER_RESTE (S1, T1) ; end if ;
if T2_DEFINI then COPIER_RESTE (S2, T2) ; end if ;
end FUSIONNER ;

```

الحلّ المفروض أعلاه هو قريب من الخوارزم الكلاسيكي لضم السجلات المتتالية في اللغات كلغة Fortran ، Cobol ، PL/ 1 ، والتي تحتاج إلى إدارة متحركة مكثس . وهو أكثر تعقيداً من الحل في لغة باسكال ، لأن في هذه اللغة ، تؤدي إمكانيات إستعمال المكثس (بالجمع) المرتبطة بالسجلات مع إعدادها وتصغيرها عند الفتح ، إلى جعل المؤثرات T1-DEFINI و T2-DEFINI غير مفيدة .

هكذا ، فالحلّ Ada أعلاه هو أكثر شمولية لأنه يسمح باستعماله مع تركيبة مختلفة لا تقمّ مطلقاً بلوغاً إلى المكثس .

13.2.2.3 إستعمال إجراء الضم للسجلات المتتالية الضميمة

```

with FUSIONNER, SEQUENTIAL_IO ;
...
declare
package SEQ_IO_INT is new SEQUENTIAL_IO (INTEGER) ;
use SEQ_IO_INT ;
procedure FUS_INT_CROIS is new FUSIONNER (INTEGER, SEQ_IO_INT) ;
procedure FUS_INT_DECROIS is new FUSIONNER (INTEGER, SEQ_IO_INT, "<" ) ;

F1, F2, F3 : SEQ_IO_INT.FILE_TYPE ;

begin
OPEN (F1, IN_FILE, "SUITE # 1", "DY1") ; -- : DY1 الأسطوانة 1 -
OPEN (F2, IN_FILE, "SUITE # 1", "DY2") ; -- : DY2 الأسطوانة 2 -
CREATE (F3, NAME => "AUX") ; -- : ضمتها ،
-- OUT_FILE

FUS_INT_CROIS (F1, F2, F3) ;
CLOSE (F1) ; CLOSE (F2) ;
OPEN (F1, IN_FILE, "SUITE # 2", "DY1") ;
RESET (F3) ; -- CLOSE (F3) ; OPEN (F3, IN_FILE, "AUX")
CREATE (F2, OUT_FILE, "FUSION", "DY1") ;
FUS_INT_CROIS (F1, F3, F2) ;
CLOSE (F1) ; CLOSE (F2) ; DELETE (F3) ;
-- DY1 : FUSION = FUSION (DY1:SUITE # 1, DY2:SUITE # 1, DY1:SUITE # 2)

-- إستعمال ممكن لـ FUS-INT-DECROIS بنفس الشكل
end ;

```

ملاحظة

نفترض هنا إن الملاحظة FORM تستخدم لتعيين ناقل محيطي ، حسب إتفاقات عدة أنظمة للميكروحاسبات .

13.2.2.4 إستعمال إجراء الضم لسجلات أسماء مباشرة

```
with FUSIONNER, DIRECT_IO ;
...
declare
  subtype NOM is STRING (1..30) ;
  package DIR_IO_NOM is new DIRECT_IO (NOM) ;
  use DIR_IO_NOM ;
  procedure FUS_NOM_CROISS is new FUSIONNER (NOM, DIR_IO_NOM) ;
  -- قياسات ضمنية على
  --
  -- READ, END_OF_FILE, WRITE على DIR_IO_NOM
  NC, NP, LG : FILE_TYPE ;
begin
  OPEN (NC, IN_FILE, "NOMS_COMMUNS", "/usr") ;
  OPEN (NP, IN_FILE, "NOMS_PROPRES", "/usr") ;
  CREATE (LG, OUT_FILE, "LEXIQUE_GENERAL", "/usr") ;
  FUS_NOM_CROISS (NC, NP, LG) ;
  CLOSE (NC) ; CLOSE (NP) ; CLOSE (LG) ;
end ;
```

ملاحظة : يفترض أن يستخدم المتغير FORM لتعيين عقدة النظام التراتبي للسجلات ، مثلاً حسب الإتفاقات في النظام UNIX . نلاحظ الفائلة من التحميل الزائد للأوامر DIRECT-IO و SEQUENTIAL-IO اللذين لهما نفس الأثر .

13.2.2.5 ضم جداول مجردة من الفقرات بعلة مفاتيح نفترض المواصفة الجزئية لجدول منظم ، وحلة من مكتبة البرامج :

```
generic
type ELEM is private ;
type CLE is private ;
with function "<" (X, Y : in CLE) return BOOLEAN is <> ;
package GEN_TABLE is
type TABLE (SIZE : NATURAL) is limited private ;
procedure AJOUTER (T : in TABLE ; C : in CLE ; E : in ELEM) ;
procedure RETIRER (T : in TABLE ; C : in CLE) ;
procedure CHERCHER (T : in TABLE ; C : in CLE ; E : out ELEM) ;
procedure INIT_LECT (T : in out TABLE) ;
procedure LIRE_SV (T : in TABLE ; C : out CLE ; E : out ELEM) ;
function FIN_DE_LECTURE (T : in TABLE) return BOOLEAN ;
```

```

private ...
end GEN_TABLE;
with FUSIONNER, GEN_TABLE;
...
GESTION_CHENIL:
declare
  type RACE is (BEAGLE, BERGER, BRAQUE, ...);
  type INFO_CHIEN is ...;
  type REF_CHIEN is
    record
      NOM : STRING (1..12);
      R : RACE;
      AGE : range 1 .. 15;
    end record;
  type CHIEN is
    record
      REF : REF_CHIEN;
      INFO : INFO_CHIEN;
    end record;

  function INF_CHIEN (C1, C2 : REF_CHIEN) return BOOLEAN;
package CHENIL is new GEN_TABLE (INFO_CHIEN, REF_CHIEN, INF_CHIEN);
use CHENIL;
BERGER_ALLEMAND : TABLE (150);
BERGER_BELGE : TABLE (50);
BERGER : TABLE (200);

function INF_CHIEN (C1, C2 : REF_CHIEN) return BOOLEAN is
begin -- مقارنة مع ثلاثة مفاتيح NOM, RACE et AGE

  if C1.NOM < C2.NOM then return TRUE; -- كتابة أخرى ممكنة --
  elsif C1.NOM > C2.NOM then return FALSE; -- return
  elsif C1.RACE < C2.RACE then return TRUE; -- (C1.NOM < C2.NOM)
  elsif C1.RACE > C2.RACE then return FALSE; -- or (C1.NOM = C2.NOM)
  elsif C1.AGE < C2.AGE then return TRUE; -- and ((C1.RACE < C2.RACE
  else return FALSE; -- or (C1.RACE = C2.RACE)
  -- and (C1.AGE < C2.AGE))

  end if;
end INF_CHIEN;

procedure WRITE (T : in TABLE; C : in CHIEN) is
-- إجراء تكييف من أجل ملء جدول معين مع احتياج متطلبات FUSIONNER
begin
  AJOUTER (T, C.REF, C.INFO);
end WRITE;

procedure READ (T : in TABLE; C : out CHIEN) is
-- تكييف من أجل القراءة مع FUSIONNER

```

```

begin
  LIRE_SV (T, C.REF, C.INFO) ;
end READ ;

procedure FUS_CHENIL is
  new FUSIONNER ( ELEM => CHIEN,
                  STRUCT => TABLE (200),
                  "<" => INF_CHIEN,
                  FIN_DE_LECTURE => FIN_DE_LECTURE ) ;

begin -- GESTION_CHENIL
  -- ملء الجدولين BERGER-BELGE و BERGER-BUEMAND
  -- جمع ذين الجدولين في BERGER
  INIT_LECT (BERGER_ALLEMAND) ;
  INIT_LECT (BERGER_BELGE) ;
  FUS_CHENIL (BERGER_ALLEMAND, BERGER_BELGE, BERGER) ;
end GESTION_CHENIL ;

```

13.2.2.6 ملاحظات عامة على هذا المثل

- مراقبة أحادية الإنشاءات S1 و S2 لم تتم ، لأنه في أغلب الحالات ستؤدي إلى ثمن تنفيذ غير مفيد . ولعدم مجازاة الإنشاءات الصحيحة بشكل مؤكد ، يمكن إضافة متغير الشمولية الإضافي : **CONTROLER : BOOLEAN** الذي يتطلب هذه المراقبة . يلزم إذا نقل الحالات الاستثنائية للإشارة إلى حالات الأخطاء .

- تعريف السجلات (أو الإنشاءات) للمسؤولة عن إطلاق الاستثناء ليس ممكناً إلا إذا قمنا بتوليد أمثلة محلّدة للإنشاءات الثلاثة S1, S2, S3 . عمليات التوليد المتعددة هذه هي باهظة الثمن على حساب المكان في الذاكرة ، ولا نستطيع إختيار هذا الحل قبل أن نكون على علم بسلوك المصروف .

13.2.3 مثال رقم 2 : معالجة السجلات المتتالية المؤشرة

13.2.3.1 تقديم المثل

الرمز **DIRECT-IO** تقدم صيغة جديدة لتصوير سجل متتالي مؤشر بواسطة الموقع . ويتعامل المبرمجون في الإدارة كيف بإمكانهم إنشاء سجلات متتالية مؤشرة على مفاتيح غير رقمية مع هذه الرزمة . المثال التالي يعرض رزمة عامة تصف النوع المجرد « متتالي مؤشر عام » باستعمال **DIRECT-IO** .

تعرض أنظمة إدارة السجلات الكلاسيكية النموذج التالي :

- السجل المتتالي المؤشر هو مجموعة من الفقرات ، مؤلفة من مفتاح ومن وصف .

- تعرّف المفاتيح بشكل مُوحّد الفقرة ، ويمكن أن تكون من أي نوع عُلّد . الأوصاف يمكن أن تكون من نوع بسيط أو بنيوي معين ، وبشكل خاص من نوع فقرة مع متغيرات (فقرات بطول متحول) .

- الأوامر التالية هي معروضة بشكل عام .
إنشاء ، تدمير (détruire, créer) ، لتحديد مدة حياة السجل .
فتح ، إغلاق (fermer, ouvrir) ، لتحريف مهمة الاستعمال والسماح ببعض عمليات المراقبة .

(ajouter, supprimer, modifier) إضافة ، إلغاء ، تعديل فقرة بمفتاح معين ، للاستيفاء اليومي للسجل .

البحث (chercher) للقراءة ببلوغ مباشر لفقرة بمفتاح معين ، بداية قراءة ، قراءة التالي ، نهاية القراءة ، لتأمين قراءة متتالية للسجل .

- الإنشاء الأولي للسجل : يمكن أن يتم بواسطة الأمر إضافة (ajouter) . أغلب أنظمة السجلات تقُدّم أمراً خاصاً يفرض استقبال المفاتيح المتصاعدة .

13.2.3.2 تحليل المسألة

برجة النموذج السابق بواسطة الرزمة DIRECT-IO تخلق مشكلة الربط بين المفتاح والموقع . وهناك حالتان يمكن أن تؤخذوا بعين الاعتبار :

- في الحالة التي يكون فيها المفتاح هو موقع الفقرة ، فأوامر DIRECT-IO ستكون مستعملة مباشرة إذا لم تكن المشكلة موجودة (مشكلة الربط بين المفتاح والموقع) بواسطة إلغاء الفقرة . هكذا إذا كانت الرزمة DIRECT-IO مجهزة بإمكانية قراءة فقرات غير متتالية (بالمعنى المجرد حسب مفهوم الموقع) ، فهي لا تقدم أية وسيلة لمحو الفقرة : تتشكّل « الشجوب » من فقرات غير محللة . إضافة لذلك ، فإن DIRECT-IO لا يقُدّم أبداً الوسائل الملائمة لاكتشاف قراءة فقرة غير مكتوبة : لا يعني ذلك بوضوح إطلاقي DATA-ERROR (هذا سيكون مقبولا) ، إضافة لذلك ، فمراقبة DATA-ERROR هي إختيارية (وهذا ليس مقبولا : تندثر جميع فوائد اللغة) . يجب إذاً ، بالنسبة لمشكلتنا ، توقع اعتماد وسيلة واضحة لتمييز الفقرات الموجودة والفقرات الغائبة . وهذا يمكن أن يتم بواسطة مؤشرات ثنائية ، موضوعة في الفقرات أو مُشحّدة في جدول .. استعمال الجدول الثنائي هو مفضل لزيادة فعالية الأعداد ، ولكنه يخلق مشكلة حول تكبير السجل . في جميع الحالات ، يجب معرفة العدد الأقصى للفقرات عند إنشاء السجل ، لاعداد المؤشرات السابقة .

- في الحالة التي لا يكون فيها المفتاح هو موقع الفقرة ، يجب إنشاء وصلة (ربط) بين المفتاح والموقع ، مما قد يتم بواسطة رزمة GINDEX . هذا الأخير يمكن أن يستعمل تقنيات مختلفة للبحث في جدول . بالرغم من إمكانيات التقسيم والشمولية الجاهزة في آدا ، فالمستعمل يمكن ، إذا رغب بذلك ، باختيار تقنيات مخصصة لذلك . من جهة أخرى ، فالرزمة GINDEX يجب أن تستعمل سجلاً خاصاً بها ، مختلفاً عن السجل الذي يحتوي على الأوصاف ، لتأمين دوام المعلومات . ليس ضرورياً ذكر المفاتيح في سجل الأوصاف ، إلا لأسباب تكاملية ، عند التدمير المفاجيء للدليل (index) مثلاً . لجعل فصل السجل المنطقي إلى سجلين فيزيائيين واضحين بالنسبة للمستعمل ، سنستعمل رزمة لإنتاج اسمين داخليين من خلال إسم منطقي مقدم من قبل المستعمل . للسجل المتسالي المؤشر الذي أوجده .

البرامج التالية تقدم رزمة تنشئ نموذجاً لسجل متساو مؤشر مع مفتاح جزأ مختلف ، وأمثلة على الاستعمال . لعدم إطالة هذه الأمثلة ، سنعرض فقط لمواصفات وحدات البرنامج المستعملة .

13.2.3.3 مواصفة الرزمة الأصلية

SEQUENTIAL-INDEXE

```
with DIRECT_IO, GINDEX ;
generic
  type CLE is private ;
  type DESCRIPTION is private ;

package SEQUENTIAL_INDEXE is
  type SEQ_IND is limited private ;
  type T_MODES is (MISE_A_JOUR, CONSULTATION, LISTAGE) ;
  procedure CREER (FICHIER : in out SEQ_IND ;
    NOM : in STRING ; FORME : in STRING) ;
    -- خلق سجل
    ETAT = CREATION ;
  procedure OUVRIR (FICHIER : in out SEQ_IND ; MODE : T_MODES ;
    NOM : in STRING ; FORME : in STRING) ;
    -- بدء دورة الاستعمال
    ETAT = MODE ;
  procedure FERMER (FICHIER : in out SEQ_IND) ;
    -- إنهاء دورة الاستعمال
    ETAT = INConnu ;
  procedure DETRUIRE (FICHIER : in out SEQ_IND) ;
    -- انتهاء دورة استعمال وتحرير الركن
    ETAT = INConnu ;
  procedure UTILISER (FICHIER : in out SEQ_IND ; MODE : in T_MODES) ;
    -- تسمح بتغيير طريقة الاستعمال
  procedure AJOUTER (FICHIER : in out SEQ_IND ;
    C : in CLE ; D : in DESCRIPTION) ;
    -- استثناء في حال CLE-PRESENTE
```

```

procedure SUPPRIMER (FICHIER : in out SEQ_IND ; C : in CLE) ;
-- استثناء CLE_ABSENTE في حال :
procedure MODIFIER (FICHIER : in out SEQ_IND ;
C : in CLE ; D : in DESCRIPTION) ;
-- Exception si CLE_ABSENTE
procedure CHERCHER (FICHIER : in SEQ_IND ;
C : in CLE ; D : out DESCRIPTION) ;
-- Exception si CLE_ABSENTE
procedure LIRE_SUIV (FICHIER : in out SEQ_IND ;
C : out CLE ; D : out DESCRIPTION) ;
-- Exception si FIN_LECTURE
function FIN_LLECT (FICHIER : in SEQ_IND) return BOOLEAN ;
-- انتهاء القراءة للتالية :
-- استثناءات تتجها وتبشها الرزمة DIRECT-IO
-- NAME_ERROR,
-- USE_ERROR,
-- DATA_ERROR,
-- DEVICE_ERROR.
-- استثناءات تتجها هذه الرزمة :
ERREUR_ETAT : exception ;
CLE_ABSENTE : exception ;
CLE_PRESENTE : exception ;
FIN_LLECTURE : exception ;

private
-- وصف آلية تغير الحالات في الجدول 2.
type ETAT is (INC, CREA, MAJ, CONS, LIST) ;
type IC is new GINDEX (CLE) ;
type SEQ_IND is
record
INF : FILE_TYPE ; -- سجل قسم الإعلام
IND : IO_INDEX ; -- دليل منسوب إلى السجل
E : ETAT := INC ;
end record ;
end SEQUENTIEL_INDEXE ;

```

Etat primitive	INC	CREA	MAJ	CONS	LIST
CREER	CREA	x	x	x	x
OUVRIR (MODE)	MODE	x	x	x	x
FERMER	x	INC	INC	INC	INC
DETRUIRE	x	x	INC	x	x
UTILISER (MAJ)	x	MAJ		MAJ	MAJ
UTILISER (CONS)	x	CONS	CONS		CONS
UTILISER (LIST)	x	LIST	LIST	LIST	
AJOUTER	x			x	x
SUPPRIMER, MODIFIER	x	x		x	x
CHERCHER	x	x			x
LIRE_SUIV FIN_LECT	x	x	x	x	

الحالة الأولية هي ' INC . الخانة (etat / primitive) تعني إن السجل الموضوع في الحالة المعنية يصبح في الحالة المشار إليها بواسطة الخلية إذا قمنا بتطبيق الأمر عليه . الحالة البيضاء تناسب حالة ساكنة ، و X تدل على خطأ (حالة إستثنائية ERREUR-ETAT) .

جدول 2 - قواعد تغيير حالات الرزمة

13.2.3.4 استعمال الرزمة SEQUENTIEL-INDEXE لإدارة المكتبة

```

with SEQUENTIEL_INDEXE, DIRECT_IO ;
package body BIBLIO is
  type CLE is ... ;
  type LIVRE is ... ;
  type REFERENCE is
    record
      C : CLE ;
      L : LIVRE ;
    end record ;

  package SEQUENTIEL is new DIRECT_IO (REFERENCE) ;
  package SEQ_INDEXE is new SEQUENTIEL_INDEXE (CLE, REFERENCE) ;
  use SEQUENTIEL, SEQ_INDEXE ;

  procedure CREER (SOURCE, PERMANENT, ARCHIVE, FORME : in STRING) is
  -- SOURCE : المجلد التالي المؤشر PERMANENT : إطلافاً من السجل التالي
  -- تعيد قراءة PERMANENT وتصنيف كل ما كتب في سجل مثال
  -- ARCHIVE :
  S : FILE_TYPE ;           -- Source : مصدر
  P : SEQ_IND ;             -- Permanent : مستمر
  A : FILE_TYPE ;           -- Archive : أرشيف
  REF : REFERENCE ;
begin
  -- : P : مخلق -
  OPEN (S, IN_FILE, SOURCE, FORME) ;
  CREER (P, PERMANENT, FORME) ;
  while not END_OF_FILE (S) loop
    READ (S, REF) ;
    AJOUTER (P, REF.C, REF.L) ;
    -- قد تحدث الاستثناء CLE-PRESENTE في حال كان SOURCE يحتوي مؤشرين على نفس الملتاح .
  end loop ;
  -- إعادة قراءة P وحفظ في A -
  CLOSE (S) ; OPEN (A, OUT_FILE, ARCHIVE, FORME) ;
  UTILISER (P, LISTAGE) ;
  SET_INDEX (A_INDEX (A, SIZE (A + 1))) ;
  -- تسمح بالكتابة التالية فوراً :
  while not FIN_LLECT (P) loop
    LIRE_SUIV (P, REF.C, REF.L)
    WRITE (A, REF) ;
  end loop ;
  CLOSE (A) ; FERMER (P) ;
end CREER ;

-- ... BIBLIO : أخرى من
end BIBLIO ;

```

13.2.3.5 Spécification des paquetages utilisés تخصيص الرزم المستعملة

with DIRECT_IO ;

generic

type CLE is private ;

package GINDEX is

- إدارة متخصصة للدليل من المفاتيح من أجل تحقيق سجلات متتالية مؤشرة

- هذه الرزمة تستلزم عيطة متيناً ، ولا تقوم بأي فحص جيد التطبيق

. type INDEX is limited private ;

type POS is DIRECT_IO.POSITIVE_COUNT ;

procedure CREER (I : in out INDEX ; NOM : in STRING ; FORME : in STRING) ;

-- Création d'un index abstrait I associé au fichier externe NOM, FORME

procedure OUVRIER (I : in out INDEX ; NOM : in STRING ; FORME : in STRING) ;

-- فتح دليل مجرد I مرتبط بالسجل الخارجي (استغناء) NOM-FORME

procedure FERMER (I : in out INDEX) ;

procedure DETRUIRE (I : in out INDEX) ;

procedure POSITION (I : in INDEX ; C : in CLE ;

ABSENT : out BOOLEAN ; P : out POS) ;

-- إذا كانت C تنتمي إلى I فإن : FALSE = ABSENT : P = وضع C في INDEX

-- في الحالة العكسية : TRUE = ABSENT : P = الوضع الذي يجب أن تكون فيه C في INDEX

procedure NOTER (I : in out INDEX ; C : in CLE ; P : in POS) ;

-- سابق : C تنتمي إلى I و P = وضعاً حراً

-- ناتج : موضع C في I في الوضع P

procedure UTTER (I : in out INDEX ; P : in POS) ;

-- سابق : يوجد مفتاح في I في الوضع P

-- ناتج : الوضع P هو حر في I

procedure DEBUT_PARCOURS (I : in out INDEX) ;

-- إصدار قراءة متتالية للدليل FINPARCOURS

function FIN_PARCOURS (I : in INDEX) return BOOLEAN ;

-- مجهول : لم يعد هناك من زوج (C, P) للتسليم في السياق الفاعل على I

procedure SUIVANT (I : in out INDEX, C : out CLE ; P : out POS) ;

-- سابق : FIN - PARCOURS (I) ، سياق فاعل

-- ناتج : (C, P) الزوج التالي الذي لم يتم تسليمه بعد

private

- يمكن اعتماد العديد من الطرق لتمثيل الدليل . كذلك يمكننا استعمال السجل الخارجي

- المرتبط بالدليل بأشكال مختلفة :

- شحن كلي في الذاكرة المركزية خلال OUVRIER ، شحن على مقاطع أو صفحات ، الخ .

end GINDEX ;

generic

CIBLE : in SYSTEM.SYSTEM_NAME ; -- نظام تشغيل مُستعمل

package NOM_FICHIER is

MAX_NOM : INTEGER ; -- CIBLE إسماً من أجل

procedure BINOMER (NOM : in STRING ;

NOM_IND, NOM_INFO : out STRING) ;

- تصنع اسمي سجلات تبدأ بأصلاحات CIBLE ، انطلاقاً من الاسم الذي يعطيه المستخدم ،

من أجل التسمين INDEX و INFORMATION في السجل التالي للوشر

```

function NOM_INDEX (NOM_INFO : in STRING) return STRING ;
-- BINOMER : NOM-INFO من انطلاقا ، تبعاً لأصلاحات ؛
end NOM_FICHIER ;

```

13.2.3.6 عمل الرزمة SEQUENTIEL-INDEXE

```

with DIRECT_IO, INDEX, NOM_FICHIER ;
package body SEQUENTIEL_INDEXE is
package NOM_F is new NOM_FICHIER (UNIX_V7) ;
package INFO is new DIRECT_IO (DESCRIPTION) ;
-- يصرح عن الدليل في قسم التخصيص --
use INFO, NOM_F, IC ;
type PRIMITIVE is (CRE, OUV, FER, DES, MAJ, CONS, LIST, AJOU, MOD, CHER,
LEQ) ;
-- مساعدات تستعملها أصليات معالجة السجلات --
A : BOOLEAN ;
P : DIRECT_IO.POSITIVE_COUNT ;
procedure TETAT (P : in PRIMITIVE ; E : in out ETAT) is separate ;
-- اختبار لصحة العملية وإبطال احتمالي للحالة حسب الجدول 1 --

procedure CREER (F : in out SEQ_IND ;
NOM : in STRING ; FORME : in STRING) is
N_IND, N_INF : STRING (MAX_NOM) ;
begin
BINOMER (NOM, N_IND, N_INF) ;
TETAT (CRE, F.E) ;
CREER (F.IND, N_IND, FORME) ; -- خالق دليل --
CREATE (F.INF, INOUT_FILE, N_INF, FORME) ;
-- خالق سجل DESCRIPTIONS
end CREER ;

procedure OUVRIER (F : in out SEQ_IND ; MODE : in T_MODES ;
NOM, FORME : in STRING) is
N_IND, N_INF : STRING (MAX_NOM) ;
begin
BINOMER (NOM, N_IND, N_INF) ;
TETAT (OUV, F.E) ;
OUVRIER (F.IND, N_IND, FORME) ; -- فتح دليل --
OPEN (F.INF, INOUT_FILE, N_INF, FORME) ;
-- فتح سجل DESCRIPTIONS
end OUVRIER ;

procedure FERMER (F : in out SEQ_IND) is
begin
TETAT (FER, F.E) ;
FERMER (F.IND) ; -- إغلاق دليل :
CLOSE (F.INF) ;
end FERMER ;

```

```

procedure DETRUIRE (F : in out SEQ_IND) is
begin
    TETAT (DES, F.E) ;
    DETRUIRE (F.IND) ;      INDEX - استرجاع للكان
    DELETE (F.INF) ;        استرجاع مكان سجل DESCRIPTIONS
end DETRUIRE ;

    procedure UTILISER (F : in out SEQ_IND ; MODE : in T_MODES) is
    begin
        case MODE is
            when MISE_A_JOUR => TETAT (MAJ, F.E) ;
            when CONSULTATION => TETAT (CONS, F.E) ;
            when LISTAGE => TETAT (LIST, F.E) ;
        end case ;
    end UTILISER ;

procedure AJOUTER (F : in out SEQ_IND ; C : in CLE ;
                    D : in DESCRIPTION) is
begin
    TETAT (AJOU, F.E) ;
    POSITION (F.IND, C, A, P) ;
    if not A then raise CLE_PRESENTE ; end if ;
    NOTER (F.IND, C,P) ;
    WRITE (F.INF, D, P) ;
end AJOUTER ;

procedure SUPPRIMER (F : in out SEQ_IND ; C : in CLE) is
begin
    TETAT (MOD, F.E) ;
    POSITION (F.IND, C, A, P) ;
    if A then raise CLE_ABSENTE ; CF3end if ;
    OTER (F.IND, P) ;
end SUPPRIMER ;

procedure MODIFIER (F : in out SEQ_IND ; C : in CLE ;
                    D : in DESCRIPTION) is
begin
    TETAT (MOD, F.E) ;
    POSITION (F.IND, C, A, P) ;
    if A then raise CLE_ABSENTE ; end if ;
    WRITE (F.INF, D, P) ;
end MODIFIER ;

procedure CHERCHER (F : in SEQ_IND ; C : in CLE ;
                    D : out DESCRIPTION) is
begin
    TETAT (CHER, FICHIER.E) ;
    POSITION (F.IND, C, A, P) ;
    if A then raise CLE_ABSENTE ; end if ;
    READ (F.INF, D, P) ;
end CHERCHER ;

```

```

4.
procedure LIRE_SUIV (F : in SEQ_IND ; C : out CLE ;
                    D : out DESCRIPTION) is
begin
    TETAT (LEC, FICHER:E) ;
    if FIN_PARCOURS (F.IND) then raise FIN_Lecture ; end if ;
    SUIVANT (F.IND, C, P) ;
    READ (F.INF, D, P) ;
end LIRE_SUIV ;

function FIN_Lect (F : in SEQ_IND) return BOOLEAN ;
begin
    TETAT (LEC, F.E) ;
    return FIN_PARCOURS (F.IND) ;
end FIN_Lect ;
end SEQUENTIEL_INDEXE ;

```

13.2.3.7 مناقشة المثال

من الممكن أن يتم تنفيذ أوالية إنشاء سجلات متتالية مؤشرة بواسطة أمر خاص للكتابة المتتالية (كما هو الحال في عدة أنظمة) ، بسهولة وبطريقة قريبة من الطريقة الموجودة . إضافة لذلك ، فعالية المفاتيح المنظمة يمكن أن تعالج بإدخال مؤشر المقارنة في متغيرات أصولية الرزم SEQUENTIEL-INDEXE و GINDEX .

المثال السابق يبرهن بوضوح إن الرزمة المحددة DIRECT-IO لا تكفي لمعالجة مسألة السجلات بتنظيم متتال ومؤشر : إدارة المؤشر هي ضرورية . الرزمة التي نعرضها تبدو كافية من الناحية العملية ، ولكنها ستكون غير فعالة عند التنفيذ ، بسبب عدة نداءات لاجراءات كالتالي يحتويها كل أمر . التحسين المهم سيكون ممكناً إذا لم نكن قد إستعملنا الرزمة DIRECT-IO ، ولكن فقط الرزمة LOW-LEVEL-IO !

الرزمة DIRECT-IO تقدم خدمات سيئة لمسائل الإدارة التي تستعمل التأثير غير الرقمي . ولكن في حالة التأثير حسب الموقع ، فإن غياب أمر محو الفقرة يمنع إستعمال الرزمة مباشرة ، ويتطلب تركيبات معطيات إضافية .

إستعمال الشمولية (الأصولية) لتعريف نوع فقرات السجل يمنع هذا الأخير من احتواء فقرات بطول متحول : نوع الفقرات المقدمة عند توليد DIRECT-IO يجب أن يكون إلزامياً ، ويجب أن تكون جميع الفقرات بنفس المتحول . هذا التضييق هو مفيد للأنظمة ببلوغ مباشر ، ولكنه غير مجدي بالنسبة للتنظيمات المتتالية فقط .

فلنشير في النهاية الى هذا التفصيل : الصعوبات المرتبطة بمعالجة الأسماء الخارجية . في المثال السابق حصرنا هذه الصعوبات في الرزمة NOM-FICHER ، التي تتعلق بالنظام الهدف بالنسبة للاتفاقات حول النحو وطول الأسماء الخارجية . وبالإمكان تفادي

هذه الصعوبات بشكل كبير فيما لو كانت الرزمة DIRECT-IO تنقل نوعاً مجزئاً خاصاً. للامضاء الخارجية للسجلات (النوع STRING هو ذو استعمال غير عملي ، لأنه يجب معرفة طول السلسلة لاستعمالها) . هذا الحل سيسمح بمعرفة جيدة للعلاقات التبعية للنظام المهدف . مع النظام الحالي ، فإن تبعية النظام الذي يستعمل DIRECT-IO بالنسبة للنظام المهدف CIBLE ، هي مركبة من إسمين ، مما يخلق مشاكل ناتجة عن إمكانية نقل البرنامج من مكانة إلى أخرى .

هكذا ففي المثال السابق افترضنا إن المتغير FORM يمكن أن يكون نفسه بالنسبة للقسم INDEX والقسم DESCRIPTION ، الذي لا يعتبر شيئاً جيداً إذا رغبنا به ، مثلاً ، وضع الأقسام على نواقل مختلفة .

13.2.4 مستوى النصوص

كما بالنسبة للسجل المتتالي أو المباشر ، فالنص هو مسجل يمثل بواسطة وحدتين ، النص المجرد المعالج بواسطة برنامج ، والنص الخاص ، الممثل على ناقل خارجي بطريقة تتعلق بالنظام المهدف . الوصلة بين النص المجرد والنص الخاص تتم بواسطة نفس الأوامر كما بالنسبة لبقية السجلات . تنقل تختلف الأوامر المستعملة في إدارة النص بواسطة الرزمة TEXT-IO ، التي تنقل أيضاً رزماً أصلية للمداخل - المخرج من أنواع رقمية أو مرقمة ، من التي يجب توليدها في عدد من النماذج يعادل الأنواع المختلفة . النص المجرد هو سلسلة من الصفحات ، متبوعة بإنهاء للنص (نهاية السجل) ؛ الصفحة هي سلسلة من الأسطر ، متبوعة بنهاية الصفحة أو النص ، السطر هو سلسلة من السمات ، متبوعة بنهاية السطر ، الصفحة أو النص .

يمكن أن يستعمل النص في القرأة (الصيغة IN-file) أو في الكتابة (الصيغة OUT-FILE) ؛ يمكن عبور النص على التوالي ، وتدل المتزلفة في كل لحظة على السمة التالية المقرؤة أو المكتوبة . يُشار إلى موقع المتزلفة [CR] بواسطة ثلاثة عدادات تعادل قيمتها 1 عند فتح النص ، وتدل على أرقام الصفحات ، الأسطر والأعمدة .

هناك أولية إختيارية تسمح بمعالجة تغييرات الأسطر والصفحات أوتوماتيكياً ، فقط بالنسبة لنصوص الإخراج ، بالإشارة إلى الطول الأقصى للأسطر أو الصفحات . وبالإتفاق ، عندما تكون هذه الأعداد القصوى معادلة لصفر (عند الفتح) ، فهذا يدل على إن الأسطر أو الصفحات ليست محلوذة : يجب وبشكل واضح ، إستعمال الإجراءات NEW-PAGE و NEW-LINE لكتابة مؤشرات الانهاء .

يمكن للنص عند الإدخال قراءته بواسطة واحد من 12 إجراء GET ، والتي تمتاز بأحد الأشكال التالية :

procedure GET (FILE : in FILE_TYPE ; ITEM : out T) ;
procédure GET (ITEM : out T) ;

ITEM تستقبل قيمة من نوع T ، يمكن أن تعني نوعاً رمزياً ، سلسلة ، نوعاً رقمياً (صحيحاً ، حقيقياً بفاصلة ثابتة أو متحركة) أو مرقباً . في حالة الأنواع الرقمية. أو المرقمة ، نحصل على القيمة بعد التحليل اللغوي ونحويل سلسلة تبدأ من موقع بالمنزلة الحالي . بالنسبة للأنواع الرقمية ، فإن الطول الأقصى للسلسلة المطلوب تحليلها يمكن أن يُشار إليه بواسطة مُتغيّر إضافي WIDTH .

نفس الشيء ، يوجد عدد من الإجراءات PUT ، التي تُضيف ، من خلال المنزلة الجارية ، سلسلة نحصل عليها بواسطة تحويل محتمل ، وبالأخذ بالحسبان المتغيرات الاختيارية مثل WIDTH ، BASE (صحيح) أو FORE ، AFTER ، EXP / حقيقية) .

وفي النهاية ، بالنسبة للأنواع الرقمية أو المرقمة ، فإن أواليات التحويل والتحليل اللغوي هي موجودة بالتصريف بواسطة إجراءات أخرى GET و PUT ، التي تعمل على سلاسل بدلاً من سجلات النصوص .

الصيغة الجديدة TEXT-IO تبدو وكأنها تقدّم لنا جميع الامكانيات المرجوة لمعالجة النصوص . المفاهيم النحوية ومواصفات الرزمة TEXT-IO تأخذ حوالي 180 سطراً ، يبقى أن نعرف إن جميع هذه الامكانيات هي بسيطة للاستعمال ، وجميعها غير إلزامي .

13.2.5 مثال رقم 3 - مداخل مخارج الفقرات النصية

13.2.5.1 تقديم المثال

إثبات طريقة استعمال الرزمة TEXT-IO في نفس نوع التطبيقات التي يمكن أن نقوم بها مع GEST LIST أو PUT LIST في 1 / PI ، DISPLAY أو ACCEPT في كويول ، المداخل - المخارج على سجلات من النصوص بلغة باسكال ستقدم فائدة قليلة لأنه من الواضح إن جميع هذه الإمكانيات هي تقريباً متساوية ، عند العبور إلى السطر تقريباً .

لقد اخترنا مثلاً قريباً عما نقوم به في لغة فورتران وكويول ، لغتان حاولت أدا أن تحلّ مكانها : قراءة وكتابة فقرات (بمعنى لغة كويول) بشكل ثابت ، ممثّل بدون فواصل بين مختلف المركبات .

تعريفات الأنواع أعلاه ، تصف التمثيل الداخلي للفقرة ، التي على السجل النصي تناسب النسق في لغة فورتران .

FORMAT (I1, I5, 20A1, F4.2)

أو في الوصف بلغة كويول .

01 ARTICLE

02 CA PICTURE 9
02 CODE PICTURE 9 (5)
02 IDENTIFICATION PICTURE A (20)
02 PRIX PICTURE 9 (5) V9
02 TAXE PICTURE 9 (2) V99.

13.2.5.2 الحل المعروض

```
with TEXT_IO ; use TEXT_IO ;
procedure EXAMPLE_3 is
  type PRIX is delta 0.1 range 0.0 .. 99999.9 ;
  type TAXE is delta 0.01 range 0.0 .. 99.99 ;
  type ARTICLE is
    record
      CA : INTEGER range 0 .. 9 ;
      CODE : INTEGER range 0 .. 99999 ;
      ID : STRING (1 .. 20) ;
      P : PRIX ;
      T : TAXE ;
    end record ;
  package ENTIER_IO is new INTEGER_IO (INTEGER) ; use ENTIER_IO ;
  package PRIX_IO is new FIXED_IO (PRIX, 5, 1) ; use PRIX_IO ;
  package TAXE_IO is new FIXED_IO (TAXE, 2, 2) ; use TAXE_IO ;
  ART : ARTICLE ;
  ASTER : constant STRING (1 .. 52) := (1 .. 52 => ' ') ;
  CARTES, DISQUE : FILE_TYPE ;
  procedure LIRE_ARTICLE (A : out ARTICLE) is
    -- تقرأ على CURRENT-INPUT سطرًا مؤلفًا من 36 سمة
    -- (النقاط العشرية هي ضمنية)
    X : INTEGER ;
  begin
    GET (A.CA, 1) ; GET (A.CODE, 5) ; GET (A.ID) ;
    GET (X, 6) ; A.P := PRIX (X)/10 ; -- : PRIX PRIX (X/10) وليس
    GET (X, 4) ; A.T := TAXE (X)/100 ; -- نفس للملاحظة
    SKIP_LINE ;
  end LIRE_ARTICLE ;
  procedure ECRIRE_ARTICLE (A : in ARTICLE) is
    -- تكتب على CURRENT-OUTPUT سطرًا مؤلفًا من 36 سمة
    -- دون نقاط عشرية للحقلين P و T
  begin
    PUT (A.CA, 1) ; PUT (A.CODE, 5) ; PUT (A.ID) ;
    PUT (INTEGER (A.P * 10), 6) ; PUT (INTEGER (A.T * 100), 4) ;
    NEW_LINE ; -- نفترض أننا بصدد أسطر غير معدودة الطول
  end ;
```

```

end ECRIRE_ARTICLE ;
procedure IMPRIMER_ARTICLE (A : in ARTICLE) is
  -- يكتب على STANDARD-OUTPUT صورة عن A
begin
  PUT (" ") ;
  PUT (A.CODE, 5) ; PUT (" ") ;
  PUT (A.ID) ; PUT (" ") ;
  PUT (A.P) ; PUT (" ") ; -- تتحدّد الأشكال عند ابتكار الزميتين
  PUT (A.T) ; PUT (" ") ; -- TAXE-IO و PRDX-IO
end IMPRIMER_ARTICLE ;
procedure TRAITER (X : in out ARTICLE) is separate ;
  -- معالجة سلعة غير عمدة

```

مثال رقم 3

استعمال الإجراءات السابقة لقراءة سجل الفقرات على البطاقات ، إعادة نسخ على سجل على الأسطوانات ، معالجة وطباعة النتائج على سجل خارجي نموذجي .

الفقرات موضوعة بين الأعمدة 10 و 45 ومكتوبة على أسطوانات بدون فراغات .

تطبع النتائج على ورق بعرض 80 عاموداً .

```

begin
  OPEN (CARTES, IN_FILE, "CR # 2") ;
  OPEN (DISQUE, OUT_FILE, "INVENTAIRE_MAJ", "TXT") ;
  -- يُفتح السجل استاندارد فمئياً وسجل الإخراج بالنقط

  SET_INPUT (CARTES) ;
  SET_LINE_LENGTH (66) ;
  NEW_LINE ; SET_COL (15) ; PUT (ASTER) ; -- السطر الأول
  while not END_OF_LINE loop -- par défaut CARTES
    SET_COL (10) ; LIRE_ARTICLE (ART) ;
    SET_OUTPUT (DISQUE) ; ECRIRE_ARTICLE (ART) ; -- DISQUE على
    TRAITER (ART) ;
    SET_OUTPUT (STANDARD_OUTPUT) ;
    SET_COL (15) ; IMPRIMER_ARTICLE (ART) ;
  end loop ;
  SET_COL (15) ; PUT (ASTER) ; -- السطر الأخير
  CLOSE (CARTES) ; CLOSE (DISQUE) ;
  SET_INPUT (STANDARD_INPUT) ;
end Exemple_3 ;

```

13.2.5.3 مناقشة هذا المثال

الإجراء LIRE-ARTICLE لا يستعمل أبداً نماذج الرزمة الأصلية

لأن السلاسل المُمثلة للقيم الحقيقية بفاصلة ثابتة يُفترض ألا تحتوي على

فاصلة عشرية . يجب إذا تأويل هذه السلاسل كأعداد صحيحة وبعد ذلك تحويلها :
النسق الثابتة مع نقطة عشرية ضمنية ليست مرغوبة بالنسبة لسجل جمع معلومات لأنها
غير مقروءة وتشكل منبعا للأخطاء .

هكذا ، - فيمكن إستعمالها لسجلات تخزين أو لسجلات مشكلة من خلال برامج
بلغة فورتران أو كوبول . في حالة النسق الثابتة بنقطة عشرية واضحة ، فإن القراءة يمكن
أن تتم باستعمال مباشر للإجراءات GET للأمثلة FIXED-IO . فلنشير إلى أن كل نوع
حققي بفاصلة ثابتة يتطلب توليد هكذا مثل : من الممكن الاستفادة من ذلك لتثبيت
المتغيرات بالغلط FOR و AFTER ، كما فعلنا .

لأنشاء فقرات مقروءة بواسطة LIRE-ARTICLE à hZbpvhw
Ecrire-ARTICLE يجب أن يكون مسبقاً بنفس الشكل كما في
LIRE-ARTICLE . وسائل معالجة السجلات بالغلط هي غير كاملة ، لأننا لا
نستطيع ، بعد تغيير السجل ، العودة إلى السجل السابق إذا لم نكن نعرف كيف كان .
فيجب أن نكتب :

التخزين - ANCIEN_FICHER := CURRENT_OUTPUT ;
SET_OUTPUT (NOUVEAU_FICHER) ;

ترميم - SET_OUTPUT (ANCIEN_FICHER) ;

ولكن هذا غير مسموح به ، لأن أنواع السجلات هي محدودة كخاصة ، وغير قابلة
للاستعمال عند التغيير . وإذا كان الاجراء معمولاً به بالنسبة للسجل الجاري ، فلا يمكن
لهذا الاجراء تعديله .

يحتاج الإجراء IMPRIMER-ARTICLE إلى ثلاثة أمثلة من الرزم الأصلية ،
لاجراء عمل شديد السهولة . الاسم «PUT» المستعمل في هذا الاجراء يناسب فعلاً أربعة
إجراءات المحلدة ، وربما خمسة إذا A.CA و A.CODE ليس لها نفس النوع الأساسي .

لقد جرى استعمال أوالية لأسطر بطول محلود في الإخراج عند استعمال
IMPRIMER-ARTICLE ولم يستعمل في Ecrire-ARTICLE . من الممكن إذاً أن
نحكم على الفائدة من هذه الأوالية التي تبدل وكأنها سيئة : فهي تساعد في تفادي
NEW-LINI ولكنها تلزم بحسبان عدد السمات الضرورية (66) . أما بالنسبة للفائدة من
المعرفة والسيطرة على طول الأسطر لجعل السطر عند الإخراج متوافق مع الجهاز ؛ فمن
الأفضل الطلب من الجهاز نفسه إجراء هذا العمل ، لجعل البرامج مستقلة عن الأجهزة .

فلنشر أيضاً إلى أن أي مُنقَح للنصوص باستطاعته إعادة تقسيم أسطر النص بترتيب أو بدون ترتيب .

استعمال إجراءات بمتغيّر عبارة عن سجل هو خطير ، كما يدل على ذلك المثال التالي :

SET_INPUT (CARTES) ;	- بطاقات
SET_LINE_LENGTH (45) ;	- خرج نموذجي
SET_COL (10) ;	- Standard output
GET (CH1) ;	- بطاقات
GET (CH5) ;	- بطاقات
NEW_LINE ;	- خرج نموذجي

إذا كان المبرمج يفكر باستعمال سجل البطاقات فقط ، فإن المتتالية السابقة هي مسموحة ولكنها غلط .

استعمال المتغيرات الصريحة يمكن أن تسمح باكتشاف SET-LINE-LENGTH (CARTE, 45) ، NEW-LINE (CARTES) غير المسموح بها . ضرورة توضيح متغيرات السجل تبرّر بدون أدنى شك وجود الدالة CURRENT-INPUT لأن هذه الأخيرة لا يمكن أن نخدم في تخزين التعريف عن السجل السابق .

13.2.6 مستوى المحيطات [MR 14.3]

يقدم هذا المستوى بواسطة الرزمة LOW-LEVEL-IO ، التي يمكن أن ترسل أنواعاً ممكنة للمحيطات والمعلومات المنقولة ، والتي تنقل الأمرين بإرسال المعلومات من البرنامج نحو المحيطات والعكس . هذه الأوامر يجب أن تكون موجودة في عدد من النماذج يعادل أنواع المحيطات المرسلّة بواسطة الرزمة .

13.3 التقييم

13.3.1 النقاط الإيجابية

اعتبار السجل وكأنه موضع ذو نوع معين ، وإدخال في هذا النوع نوع مركبات السجل ، هو النقطة الأكثر إيجابية في هذا المستوى . المثال بلغة باسكال أثبت تبسيط التصوّر ونسبة الأمان التي حملتها هذه الفكرة ، بالنسبة إلى ما كانت تقدمه جميع اللغات السابقة .

نوع السجلات يدخل الآن في صيغة البلوغ (متثال أو مباشر) ، ولا يدخل فيه إنجاز الإرسال (إدخال - إخراج) ، مما يبدوننا أنه تقدم كبير بالنسبة للمصنف السابقة .

التدقيقات الساكنة التي يمكن أن يقوم بها المصرف تتعلق باستعمال أوامر متعلقة بكل ما يجب أن يُختبر عند التنفيذ .

قدمت مجموعة الأوامر الموضوعية بتصرفنا ، كما يبدو ، كل ما هو مطلوب في مستوى بسيط . التوافق في الأوامر بين الصيغ المتتالية والمباشرة ، والمحافظة عليه في كل مرة ، يكون ذلك ممكناً ، يبدو لنا وحيداً ويختص بتبسيط ما يجب أن يعلمه المبرمج .

13.3.1.2 الانتقادات

أ - فقدان المواصفات الواضحة

هذا العيب يعود للوثائق التي درسناها ، chapter reviews المؤرخة في ديسمبر 1981 ، الممكن أن يُصحح في الصيغة النهائية [MRA] المنشورة في نفس وقت هذا التقييم . أغلب التصورات المستعملة ليست « مغلدة » إلا ضمناً ، مما يسمح بعبء تأويلات . . . المواصفة الإلزامية ، وحتى ولو كانت قليلة التجرد ، تبدلنا وكأنها ضرورية لاثبات صلاحية البرنامج الذي يستعمل الرزم المعروضة ، وهذا ما لا تقدمه الفقرات [MRA 14.1.4 ، 14.6 ، 14.18 ، 14.20] التي لا تدل إلا على التصورات النحوية .

من الصعب مثلاً تحديد ، وبشكل سهل واضح ودقيق ، ما هي تصورات السجلات الداخلية والخارجية والمفاهيم التي ترتبط بها ، مثلاً ، العلاقة التي تربطها مع حجم السجل . نفس الشيء نقوله بالنسبة للمتغيرات NAME و FORM ، كما تدل على ذلك الفقرة التالية .

ب - تبعية المحيط التنفيذي

يمكن للمبرمج أن يدرك مفهوم السجل الخارجي ، لأنه يوجد أوامر إتصال بين السجلات الخارجية والداخلية ، ومفاهيم مناسبة للمسجلات الخارجية ، مثل SIZE ، NAME ، FORM ، STANDARD-INPUT ، الخ . وهذا سيكون - بدون سيئات إذا أخذت بعض الاحترازمات لمراقبة العلاقة التبعية بين تصورات السجل الخارجي بالنسبة لمحيط التنفيذ الخاص به . وللأسف لم تكن هكذا هي الحالة بالنسبة للرزم المعروضة سابقاً .

هكذا فالمتغيرات NAME و FORM لها خصائص عامة مشتركة بين جميع محيطات التنفيذ التي ليست مميزة بخصائص خاصة بكل نظام تنفيذ . مثلاً تعيين الناقل ، نحو الاسم ، التعبير عن حقوق البلوغ . هذه المتغيرات تقلّم كسلاسل يؤدي إستعمالها إلى إزعاج المبرمج وقد تفسد جديداً إمكانية نقل البرنامج الذي يقوم بالإدخال - الإخراج . ضرورة معرفة حجم هذه السلاسل لاستعمالها ، غياب الدقة في النحو والدلالة . إذا كانت مواصفة رزم الإدخال - الإخراج . تُدقّق بشكل جيد في هذه

المعلومات ، فهذا سيسمح على الأقل باستعمالها بشكل صحيح ، ولكن هذا لا يُصلح مشكلة إمكانية النقل ، لأنه لن يكون من الممكن تمييز البرامج عن برامج أخرى لا تستغل إمكانات المحيط الخاصة .

نفكر إذاً بأنه من الأنسب إستبدال الرزمة الحالية IO-COMMON بواسطة رزمتين مختلفتين STANDARD-IO و SPECIAL-IO ، وهذه الأخيرة تُنقل بواسطة الرزمة SYSTEM . الرزمة STANDARD-IO ستحتوي على كل ما هو ضروري للعمل بالسجلات الخارجية ، بإمكانيات مشتركة بين المحيطات . هكذا مثلاً في حالة أوامر إدارة السجلات ، التي ستقدم كحمل زائد الى مختلف أنواع السجلات الداخلية : متتالية ، مباشرة ، ونصية . مفهوم الاسم الخارجي سيكون مقدماً بواسطة نوع خاص ، مع مؤثراته الممثلة بواسطة إجراءات أو دوال ، والتي ستلعب دور « الاسم الخاص الخارجي » . الربط بين « الاسم المجرد الخارجي » و « الاسم الخاص الخارجي » يمكن أن يتم بطريقة غير مرئية في البرنامج (باستعمال أو بإنشاء أسياء في حالة وجود سجلات مؤقتة) ، أو بواسطة البرنامج بسبب وجود الرزمة SPECIAL-IO . هذه الرزمة الأخيرة ستنقل نوع « الاسم الخاص الخارجي » وجميع الامكانيات الخاصة بمحيط التنفيذ وبشكل خاص تلك المعتمدة بواسطة المتغير FORM . إستعمال إحدى هذه الرزم المشار إليها بواسطة الجملة with التي تُحدد وحدة التصريف التي تقوم بالإدخال-الإخراج ، سيكون شديد الوضوح ، وهذا سيسمح بتمييز البرامج القابلة للنقل عن الأخرى .

ج - نقد التفصيل

إذا كان لأحد البرامج سجل كمتغير شكلي ، فكيف يمكن أن يُنقل هذا الأخير ؟ في رؤوس الرزم ، الصيغة هي in out بالنسبة للإجراءات CLOSE ، OPEN ، CREAT و DELETE ، وجميع الإجراءات الأخرى (بما فيها RESET) . يبدو من غير الطبيعي إستعمال هذه الصيغة الأخيرة لشيء آخر غير المهام التي تعطي قيمة خاصة للسجل ، لأن البرامج الثانوية الأخرى لها تأثير على مضمون حالة السجل . لقد رأينا من جهة أخرى (في 13.2.2.1) إن عدم الشمولية في الصيغة in out تمنع التعميمات المفيدة .

الإستثناءات المنقولة بواسطة الرزم لا تسمح باستعمال أية تقنية للترميم ، لأنه لا يوجد وسيلة لتعريف أسباب الأخطاء ، حتى ولا السجل المسبب في حال وجود عدة سجلات . إن توليد عدة نماذج عن الرزمة المستعملة لا تقدم شيئاً ، لأن جميع الإستثناءات تبعث فعلاً ، تحت عدة أسماء متشابهة - الرزمة IO-COMMON .

الإستثناءات الممكنة لا تناسب فئات كثيرة من الأخطاء ، وبشكل خاص USE-ERROR ، وهذا الاستثناء الأخير ، بالإضافة إلى NAME-ERROR

وDEVICE-ERROR ، لم يُعرف بالكامل ، والشروط التي يمكن أن يحدث فيها يُفترض أن تتعلق بالعمل . الحل الوحيد لمحاولة ترميم الأخطاء هو في حصر كل نداء للإجراء في فقرة تحتوي على مرمم للاستثناء : يبدو وكأننا نرغب بأن يقوم المبرمج بمحاولات قليلة في هذا الإتجاه .

فلنشر أخيراً إلى الفائدة القليلة حالياً في الرزمة DIRECT-IO في الأوامر SIZE وEND-OF-FILE ، لأن معناها يتعلق بعملها عند تنفيذها .

13.3.2 مستوى النصوص

13.3.2.1 النقاط الإيجابية

الابتعاد الكامل عن مفاهيم النسق ولانحة الإرسال التي كانت تعقد كثيراً الإدخال - الإخراج في فورتران ، الغول 68 أو في 1 / PL ، يبدو لنا شيئاً جيداً . فلقد قدمت لغة 68 Algol بعض الإمكانيات البسيطة ، و 1 / PL قدمت مواصفات LIST وDATA ، لتضادي إستعمال النسق ، ولكن في كلتا الحالتين كانت النتيجة تفتقد للبساطة . الاختيار الذي تم في آدا كان الأفضل إذا لم يكن من الواجب إضافة شيء جديد إلى اللغة بالنسبة للإدخال - الإخراج ، على الأقل لقبول النسق على شكل سلاسل من السمات المؤولة بانتظام ، وباستبدال لوائح الإرسال بواسطة إجراءات شمولية معقدة ، حسب مثال الإدخال - الإخراج الذي أضيف إلى Algol 60 .

الفصل في مستوى النصوص عن مستوى السجلات يبدو لنا شيئاً جيداً ، وهذا ما يظهر لنا في الصيغة السابقة .

13.3.2.2 الانتقادات

لقد أدخل مفهوم السجل النموذجي تعقيدات كثيرة لأشياء قليلة : أربعة دوال وإجراءين ، إضافة إلى صيغة جديدة إضافية لكل مثال عن الإجراءات الأخرى والدوال . إضافة لذلك ، فهذا هو عنصر عدم أمان غير مقبول ، لأن تغيير السجل بالغلط يمكن أن يتم بطريقة غير مرئية ، في رزمة منقولة ، أو بطريقة متزامنة ، في مهمة أخرى . تصحيح الصعوبة بضمن عدم الأمان لا يعتبر تقدماً .

التوازي المطلق الذي نبحث عنه في إجراءات معالجة الأعمدة ، الأسطر والصفحات ، بين تلك التي تعمل في القراءة وتلك التي تعمل في الكتابة هو إصطناعي . قراءة علامات الصفحة ، بشكل خاص ، تبدو قليلة الفائدة ، وليست ضرورية لتسمح بنسخ متطابق لنص يحتوي على علامات الصفحة . إمكانيات الفراغ في القفز عن أسطر عند الإدخال هو من نفس نوع الشمولية غير المفيدة . هل حقاً من الضروري أن يسمح الأمر بعدم قراءة سطر على سبعة من نص معين ؟

نأسف أنه عند القراءة سمة بعد سمة ، تكون مختلف علامات الصفحة والسطر « غير مرئية » من قبل الاجراء GET . بكلمة أخرى إذا قرأنا نصاً من السمات سمة بعد سمة ومن فحص قيم الدوال END«OF-LINE وEND-OF-PAGE ، فلا شيء يفصل السمة الأخيرة في السطر (أو من الصفحة) عن السمة الأولى في السطر (أو من الصفحة) التالي .

إمكانية حصر الأسطر والصفحات تبدولنا هجينة : بإمكان المبرمج أن يُعرِّفها بنفسه ويسهولة ، أو على العكس ، بإمكان الرزمة أن تفرض وسائل سهلة تذهب بوضوح بعيداً في حقل معالجة النص .

إدخال السمات في الأنواع المرقّمة يؤدي إلى أوهام إذا كان بالإمكان قراءة الأنواع المذكورة مباشرة ، كما يدل على ذلك المثال التالي أعلاه :

```
use TEXT_IO ;
package CHAR_IO is new TEXT_IO.ENUMERATION_IO (CHARACTER) ;
TEXT_IO.PUT ('A') ;      - يكتب 'A'
CHAR_IO.PUT ('A') ;      - يكتب 'A'
--
-- نشير إلى أن PUT (٩٨) قد تكون موضع التباس !
```

- فلنشير إلى أن PUT ('A') سيكون مبهماً .

هذا المثال يبدو لنا ولعمري ! الصعوبة لا تأتي من عملية الإدخال - الإخراج بل من لغة آدا نفسها ، كذلك من تلك التي نُجبرنا على تعريف فعل التوليد .

```
package INTEGER_IO_BIS is new TEXT_IO.ENUMERATION_IO (INTEGER)
```

لسبب بسيط كون اللعبة < > تحيط بالأعداد الصحيحة ، يلزم 5 صفحات « لتحديد » القسم المرئي من الرزمة REXT-IO . نجد فيها 5 أنواع ، أربع ثوابت ، 8 استثناءات ، 65 برامج ثانوية ، وأربعة رزم أصلية يرسل كل منها 6 برامج ثانوية . الاستعمال يختلف تقريباً إلى 32 عدد الأسماء المختلفة للبرامج الثانوية . تعقيد المجموع هو أمر مقلق ، ومن الممكن أن نتساءل كم سنبذلنا لوصف القسم الخاص - من هذه الرزمة .

13.3.3 في مستوى المحيطات

تعريف هذا المستوى يبدو لنا مفيداً إذا كان يسمح بجعل المفهوم الأقل إمكانية للتقل للإدخال - الإخراج ، نموذجياً ، وإذا كان يسمح بإثبات إمكانية صنع إدخال - إخراج بالمستويين الآخرين في لغة آدا .

13.3.4 لا إضافات إلى اللغة

13.3.4.1 النقاط الإيجابية

- هذا الاختيار سيكون إيجابياً إذا كان يسمح فعلياً بالوصول إلى الحالات التالية .
- تبسيط اللغة من وجهة نظر المكنة .
- تبسيط اللغة من وجهة نظر المبرمج .
- إختزال حجم العمليات .
- جعل مفهوم البرامج نموذجية ، دون منع ذلك من أن يتم بشكل آخر .
- السماح للغة بمجاراة التطور السريع للتكنولوجيا .

13.3.4.2 نقد أسباب الاختيار

علم إضافة أي شيء إلى قواعد لغة آدا لتعريف المداخل - المخارج ليس هو تبسيط بالنسبة للمبرمج . المداخل - المخارج تُشكّل قسماً من اللغة ، لأنها موصوفة في الفصل 14 من [MR] ، وليست في الملحق ، وهي تثقل بشكل كبير ما يجب على المبرمج معرفته : أربع رزم عادية وأربع رزم أصلية ، وأكثر من مئة برنامج - ثانوي مختلفة ، وحوالي 50 إسماً ، إلخ .

من وجهة نظر العامل ، يجب في نهاية الحساب إجراء التنفيذ ، وفي أغلب الحالات لا يمكن أن يكون تعريف اللغة آدا لعبة للأولاد . فالتسهيلات ، والإلغاءات ، والتنظيفات ، وعمليات الإسقاط في الحقول الأخرى ستسهّل كثيراً ، من مهمة العامل .

الاعتقاد بأن وجود رزم محلّقة هو دليل على أن مجموعات المستعملين يمكن أن تُعرف رزماً للإدخال - الإخراج وجعلها نموذجية [MB 15.1] ، هو مجرد خديعة . فقبل أي شيء يكون الاختيار موضع السؤال غير كامل ، لأن أجسام الرزم هي غائبة ، والدلائل المعطاة في [MB 15.5] على طريقة برمجتها تجعلنا نعتقد بعدم وجود فعالية ملحوظة . ومن جهة أخرى ، فالمؤلفون يعترفون ضمناً ، بأنهم لم يصلوا إلى إنتاج مداخل - مخارج كافية ، لأنهم يشجعون مستعملي اللغة على تعريف مداخل - مخارج أخرى ونجدد الإشارة هنا إلى أن موضع الإدخال - الإخراج هو الموضوع الأكثر تطوراً بين مختلف الصيغ المتتالية .

13.3.4.3 النتائج على اللغة وعلى العاملين

نظرياً ، المداخل - المخارج لا تحتاج إلى زيادة عدد الإنشاءات النحوية والدلالية للغة ، ولا إلى توسيع المعنى . من جهة أخرى ، فمصرف آدا يجب أن يقدر على إهمال وجود رزم الإدخال - الإخراج ، ومعالجتها كالأخرى . ولوقام المؤلفون بإضافة بعض

الشيء للغة آدا للسماح بكتابة القسم المرثي من رزم الإدخال - الإخراج ، لكان ذلك يتعلق ببعض السهولة في الاستعمال العام ، وعلى عكس منسق لغة الغول 68 ، التي تمتاز بعمليات الإدخال - الإخراج والتي لا يمكن أن تعمل بدونها ، لأن أية عملية أخرى غير معتمدة لهذا النوع .

إمكانية بلوغ الهدف الثاني (لا يوجد حالة خاصة في المصرف بالنسبة للإدخال - الإخراج) مشكوك بها . بإمكاننا في هذه الحالة ، التشكيك بالنتائج حول حجم وفعالية البرامج ، لأن ذلك يتعلق بحالة حيث الأصولية الشمولية ، التحميل الزائد ، مجموعات الأنواع والتحويل الضمني هي موضوعة تحت إختبار صعب .

هكذا ، فمن الضروري توليد نموذج على الأقل لكل رزمة SEQUENTIAL-IO أو DIRECT-IO لكل نوع من مركب السجل الذي نرغب بمعالجته . يلزم نموذج لإحدى الرزم الشمولية الأصلية المنقولة بواسطة الرزمة TEXT-IO لكل عائلة أنواع صحيحة (مثلاً SHORT-INTEGER ، ، LONG-INTEGER) ، لكل عائلة من الأنواع بفاصلة متحركة (مثلاً SHORT-FLOAT ، FLOAT ، LONG-FLOAT) ، لكل نوع بفاصلة ثابتة ولكل نوع مرقم ، نرغب بإجراء الإدخال - الإخراج عليه . إلى هذه الأخيرة يجب إضافة ما هو مولد أوتوماتيكياً لكل نوع [MR.3.4.P.3-7] .

وفي المجموع فإن عدد الإجراءات والدوال الموجودة في كل برنامج يرغب بإجراء إدخال - إخراج على مواضيع وأنواع مختلفة يمكن أن يصل إلى عدة مئات . من الممكن تصور عمل خاص يؤدي إلى استعمال نموذج واحد من كل إجراء ، أو استعمال الإجراءات المستعملة فعلياً ، والتي تعالج بطريقة أخرى موضوع الأصولية .

وبرأينا ، فإن الاختيار الموجود أمام العامل الأمين على المرجع - المساعد للغة آدا هو التالي : تعقيد بشكل كبير لعمله (نعرف عدم فعالية وعدم أمانة هذا النوع من المصنفات) كي يكون قادراً على معالجة رزم الإدخال - الإخراج بشكل فعال ، أو معرفة الأنواع والاستثناءات وأوامر الإدخال - الإخراج ومعالجتها بشكل خاص . وفي الحالتين ، فإن المؤلفين يقتلون إلى هدفهم ؛ الذي يقوم على تفادي تعقيد الأعمال بالنسبة للإدخال - الإخراج .

٤.٣.١٣ النتائج على استعمال المداخل - المخرجات

● في مستوى السجلات ، كون السجل ليس إنشاءً من نوع اللغة ، بنفس موضوع الجداول والفقرات ، يلزم باستعمال الشمولية ، حيث يبقى الشكل مقبولاً ، لأنه يجب فقط توليد نموذج للرزمة المطلوبة لكل نوع من المركبات ، وإستعمال تعبير مُعْمَرٌ لأسماء

أنواع السجلات . وللمشمولية سبستان كبيرتان : من جهة ، فإن شروط الأنواع يجب أن تكون مثبتة عند توليد نموذج الرزمة ، مما يعني إن جميع مركبات السجل يجب أن تكون من نفس نوع الشرط الملزم . هذا التقييد سيتم إعتباره تغيير غير مبرر بالكامل وغير مقبول بالنسبة لأغلب المبرمجين ، على الأقل في حالة السجلات المتتالية ، ويؤدي إلى اعتماد حيل ، في البرمجة ، باستعمال مواصفات العناوين والتحويلات غير المدققة بها . ومن جهة أخرى ، فإن الشمولية ليست في نفس الوقت ملزمة كثيراً ، لأنها تسمح بتعريف مبهم لسجلات المؤشرات ، الجداول وحتى السجلات : ماذا يجب أن يفعل المصنف في هذه الحالة ؟

إضافة إلى الرزم ADA التي لا تقدم أبداً نفس الامكانيات كالأنواع المحددة مسبقاً ، لا يوجد إمكانية مواصفة التمثيل . المشكلة هي في عدم وجود ، وبالتحديد بالنسبة للأنظمة المتتالية المؤشرة والانتقائية ، عمل فعال ، لجميع المسائل ، ولجميع التشكيلات ، ولجميع صيغ التشغيل . يجب إضافة وسائل تكييف إلى رزم الإدخال - الإخراج ، للسيطرة على التمثيل وتنظيم الفقرات على مختلف أنواع العتاد ، وجعل عدد العمليات الفيزيائية هو الأفضل (تحريك ذراع القراءة) ، وفرض خوارزميات (إدارة المؤشر ، دالة العنونة المثورة) ، وإعلام المبرمج بالاختيارات الجارية على محيط التنفيذ . هنا ، هذه التكييفات يجب أن تكون مهمة ، مما يتوافق قليلاً مع الرغبات في هذه اللغة ، وتفصل في خارجها .

● في مستوى النصوص ، الصعوبة تضاف إلى عدم الفعالية . لأن الإجراءات لا يمكن أن نحصل سوى على عدد متحول من المتغيرات ، يلزم نداء من نوع PUT و GET لكل قيمة يلزم إرساها . حتى إذا كان نداء الإجراء نسبياً قليل الثمن ، فإن زيادة النداءات تزيد الثمن كثيراً .

في كل ما يتعلق بالصعوبة في الكتابة ، فإن المؤلفين يتهرون منها بسرعة بقولهم إن اختيار الأسماء القصيرة يضعف هذه السيئة .

من جهة أخرى ، فإذا كانت عمليات الإدخال - الإخراج على سجلات النصوص لا تتعلق بالسجلات النموذجية ، وحسب ، ولكن تعالج إضافة لذلك لسجلات أخرى ، فيجب إما تحديد السجل المعتمد عند كل نداء للبرنامج الثانوي ، وإما استعمال إمكانية « السجل بالخلط » ، وذلك بتغيير اسم هذا السجل . الإمكانية الأولى هي صعبة ، والثانية تضيف معامل عدم أمان خطير ، والاثنتان هما غير فعاليتين . في حين سنكتب في باسكال .

نفترض التصريحات التالية |
type naturel = 0 .. **maxint** ; صحيح
 r  el = *real* ; حقيقي
var *x* : *r  el* ; *i* : *naturel* ;
 r  ultats : *texte* ; النص
 |
writeln ('*x* = ', *x* : 10 : 3, ', *i* = ', *i* : 5) ; **sortie standard** | **نـمـراج**
writeln (*r  ultats*, *x* : 15, *i* : 8) ; **التـالـج** .

في لغة آدا يجب بدلاً من ذلك كتابة

- نفترض التصريحات التالية

```
-- X : FLOAT ; I : NATURAL ;
-- use TEXT_IO ;
-- RESULTATS : TEXT_IO . FILE_TYPE ;
-- package ENTIER_IO is new INTEGER_IO (INTEGER) ;
-- package REEL_IO is new FLOAT_IO (FLOAT) ;
-- use ENTIER_IO, REEL_IO ;
SET_OUTPUT (STANDARD_OUTPUT) ;
PUT (" X = ") ; PUT (X, 10, 3) ; PUT (" , I = ") ; PUT (I, 5) ;
NEW_LINE ;
SET_OUTPUT (RESULTATS) ;
PUT (X, 2, 8, 2) ; PUT (I, 8) ; NEW_LINE ;
```

باستطاعتنا أن نفترض أن عملية توليد الرزم ENTIER-IO و REEL-IO قد تمت أوتوماتيكياً في رأس البرامج ، وبواسطة صيغة تصريف معينة مثلاً . وللأسف ، إذا كان العمل الخاص الذي نستعمله يحدّد عدة أطوال للأعداد الصحيحة والحقيقية ، فإن التعريف - الأول الأوتوماتيكي للرزم المناسبة تصبغ سيئة أكثر منه فائدة ، لأن النداء السهل مثل PUT (10) أو PUT (0.0) يصبح مبهماً في جميع الأشكال ، في المثال أعلاه ، فإن النداء PUT (I,5) ليس مسموحاً إلا لأننا نفترض إن المصروف يضع الأولية الصعبة للبرامج الثانوية المشتقة .

المثال أعلاه يستعمل جميع الاختصارات الممكنة . لا نستطيع أن نستعمل في نفس السطر من البرنامج أربعة نداءات لـ PUT بسبب استعمال « السجل بالغلط » . هذا الإستعمال ، يتطلب النداء إلى SET-OUTPUT الذي يسبق مباشرة ، لأنه لا يوجد وسيلة لمعرفة ما هو السجل بالغلط الجاري .

ستخزل الإمكانات المختلفة للتحويلات الضمنية أو الصريحة ومشتقات ابرامج - الثانوية عدد عمليات التوليد الضرورية بالنسبة للرزم INTEGER-IO و FLOAT-IO للأنواع الأساسية الجاهزة بتصريف العمل (من I إلى ثلاث بشكل عام) . وعلى العكس ، فهذا ليس ممكناً ، لا بالنسبة للأنواع بفاصلة ثابتة ، ولا بالنسبة للأنواع

المرقمة ، ويجب أيضاً في جميع الحالات توليد نموذج للرمزة الشاملة الأصلية .
FIXED-IO لكل نوع بفاصلة ثابتة التي نرغب عليها بإجراء عمليات المداخل -
المخارج ، أو للرمز الأصلية ENUMERATION-IO لكل نوع مرقم .

إمكانية قراءة وكتابة الأنواع المرقمة تؤدي إلى الحالة الوحيدة في جميع اللغات حيث
الإسم يصبح مبلوفاً خارج مدى وجوده ، ويكون أقل تقييداً . وليس مؤكداً من أن
الاسماء المختارة في البرنامج لقيم النوع المرقم ، مع الأخذ بعين الاعتبار لمختلف
الشروط المفروضة بواسطة التنازع الممكن مع الكلمات - المفاتيح أو مع الاسماء الأخرى
(هذه الشروط هي متعددة ، لأنه يجب إختراع عدد كبير من الاسماء في كل برنامج بلغة
آدا) . هي أكثر ملاءمة في موضوع الإتصالات بين البرنامج ومستهلميه . حيث
مستهلم البرنامج يرغب بكتابة out ، in ، old ، new ، end ، gogin ، يجب أن
يفرض عليه أسماء أقل طبيعية ، وإما باستعمال الامكانية المقيدة بواسطة الرزمة
ENUMERATION-IO ، وبرجة التكويد والتعرف على هذه الاسماء . إذا كانت
هذه التسهيل مفيدة للمبرمج ، وبالنسبة للاستيفاء اليومي للبرنامج) ، فهي ليست
متكيفة مع حاجات المستعمل للبرنامج المكتوب بلغة آدا .

13.4 النتائج

13.4.1 نماذج أخرى بالنسبة لمفهوم الإدخال - الإخراج

المفهوم الذي إختاره المؤلفون للغة آدا لتعريف المداخل - المخارج هو طبيعي .
ويقوم على اعتبار مفهوم الإدخال - الإخراج كبداية بين النظام التطبيقي المكتوب
بواسطة برنامج بلغة آدا وأحد السجلات الخارجية ؛ تنظيم هذا الأخير جرى إختياره
بالنسبة لمجموعة أساسية (هنا سجلات متتالية ، سجلات مباشرة وسجلات نصية) .

المفهوم الآخر ، وهو أفضل من السابق ، سيكون أيضاً ممكناً . وسيقوم على تمييز
بعض الدوال المنطقية المعتمدة بالنسبة لمختلف حلقات الإدخال - الإخراج ، وسيحتاج
إلى أربعة نماذج مختلفة للأنظمة بالنسبة لتبادل المعلومات مع نظام تطبيقي . سنقوم
بتطوير هذه الفكرة في القسم الحالي .

هذا المفهوم يأخذ بالحسبان أشكالاً مختلفة للأدوات المحيطة المستعملة في التبادل بين
الإنسان والمكنة على شكل نصوص مطبوعة ، على شكل أشكال أو أصوات .

يعتبر البرنامج التطبيقي آدا كنموذج لنظام معلوماتي موجود في محيط المستعملين .
هذا الأخير يمكن أن يتصل مع أربعة أنظمة معلوماتية ، لتخزين ، إستلام أو ترميم قيم
في محيط المستعملين ، أو لتبادل القيم مع مراكز معلوماتية بعيدة .

الفائدة في هذا التفريق يأتي من الطبيعة المختلفة للمعالجات المطلوب إجراؤها وعن

عدم إمكانية التجرد بشكل كامل عن المسافة الجغرافية . ولكن من الممكن تخزين قيم في نظام بعيد ، ولكن ، في هذه الحالة ، فإن معيار الوقت يتغير ولا يمكن بلوغ معلومة بعيدة بالسرعة التي قد نبلغها فيها إذا كانت موجودة بالقرب مباشرة : البعد يؤدي عادة إلى إدخال نظام إدارة قليل أو كثير التعقيد . يأخذ وقتنا غير قليل بالنسبة لبرونوكول الإرسال ، إرسال الرسائل ، والتنازع للبلوغ .

الأربعة أنظمة التالية ، يمكن أن نحصل على المهام التالية :

نظام تخزين يمتاز بالمهام العامة التي يمتاز بها نظام إدارة مجمع للمعطيات (S.G.B.D) . هذا النظام يجب أن يسمح ليس فقط بتخزين المعطيات ، ولكن بالتعبير أيضا عن العلاقات المنطقية بين هذه المعطيات ، وتقديم تقنيات بلوغ متخصصة . هكذا نظام يجب أن يسمح للبرامج التطبيقية بالتجرد عن التقنية المستعملة لتخزين المعطيات .

نظام الإستقبال لا يجب أن يحاول تكويد الظواهر الفيزيائية الملحوظة في محطات الإدخال . يجب أيضاً أن يتحكم بجميع القواعد الدلالية والنحوية التي يجب أن تراعي الأشكال الخارجية للمعطيات المنتظمة . وفي مستوى التصميم ، هكذا نظام يجب أن يتصرف على النحو ، الشكل ، وتنظيم الحوار مع المستعملين من البشر ' نظام الترميم الذي يأخذ على عاتقه المسائل النفسية الناتجة عن العمل والتي نظهر عندما نرغب بإرسال رسالة معينة إلى كائن بشري وعندما نرغب بأن تكون « هذه الرسالة مفهومة » .

مسائل « التقديم الجيد » للنص ، للمخطط ، للتبادل البصري - السمعي ، الخ . نظام النقل يتمتع بمهام الأنظمة الحالية التي تسمى عادة « شبكات » (مثلا ، S.N.A لـ IBM) .

لكل من هذه الأنظمة مهمة التجرد ومعالجة الأخطاء المختلفة ، مما يبرر كثيرا هذا التفريق . ولكن هذا المفهوم يبدو لنا وكأنه مستقبلي ، كما بالنسبة للمسائل التقنية التي يفترض حلها .

هكذا رؤية حول الإدخال - الإخراج ، والتي قد تبقى موجودة في يوم ما ، في 10 أو عشرين سنة ، يمكن أن تبرز إختيار المؤلفين لعدم وصف عمليات الإدخال - الإخراج في اللغة : الأنظمة المذكورة ، وفي صيغة مبسطة ، يمكن أن تعرض على البرمجيين وبشكل رزم مجهزة ، كوسائل الإدخال - الإخراج الحالية .

هكذا ، فلنلاحظ أن الامكانيات الحالية للغة أدا تبدو سيئة عند إستعمال S.G.B.D . لذا عندما نضع بتصرف المبرمج إمكانيات متقدمة لنظام S.G.B.D علائقي ، فلهذا الأخير الحق بطلب وسائل نحوية ملائمة لتحديد طريقة بلوغ المعطيات ، كما يتم ذلك في لغة لمعالجة المعطيات أو في بعض توسيعات لغة كويول .

هكذا مبرمج ، معتاد على التعبيرات من النوع :
» لكل شخص من الجنس = الذكر و العمر < 18 يحصل على مخالفة للسبب = زيادة
السرعة أفعّل ... « .

" pour toute personne de sexe = masculin et d'âge > 18 ayant reçu une
contravention pour cause = dépassement-de-vitesse faire ... "

سيجد إنشاءات للتحكم في لغة آدا ضعيفة ، حتى لو كانت متطورة بالمقارنة مع
اللغات الأخرى .

هذه الفقرة تطرح الصعوبات القصوى التي تفرضها عمليات الإدخال - الإخراج
التي تطمح بأن تكون مستعملة لوقت طويل ولعدد كبير من التطبيقات : الأربعة أنظمة
السابقة ، وبالرغم من صفتها المستقبلية ، فهي لا تناسب التطبيقات في عالم الروبوت أو
التحكم بالعمليات الصناعية .

13.4.2 توسيعات اللغة

في الغول 68 ، جرى إضافة النسق على الأقل إلى اللغة للسماح بكتابة عمليات
الإدخال - الإخراج في اللغة نفسها ، وجميع التحويلات الضمنية ، سمحت بتصريف
الإجراءات التي تستقبل عدد متحول من المتغيرات ، بأنواع مختلفة ، وبشكل غير
إصطناعي .

وفي أغلب اللغات الأخرى التي تعرض إمكانيات للإدخال - الإخراج عامة ، فإن
هذه الإمكانيات تدخل في صلب اللغة نفسها وتزيد من لائحة الإنشاءات الممكنة والتي هي
بدون فائدة بالنسبة لباقي اللغة .

وإذا كانت لغة آدا لا تسمح بتعريف عمليات الإدخال - الإخراج بشكل مقبول ،
فيجب توسيعها . فهل يجب أن تكون التوسيعات المناسبة مستعملة ؟

بإمكاننا مثلاً ، بالنسبة لعمليات إدخال - إخراج النصوص ، إدخال إمكانية لوائح
المتغيرات بطول متحول ، أو أي شيء شبيه بالصيغة المتحولة في لغة الغول 68 .

يجب أيضاً أن نطلب ماذا ستكون الفائدة من تعريف عمليات الإدخال - الإخراج
بواسطة اللغة نفسها . ففي الغول 68 ، قمنا بالبحث لوصف الإدخال - الإخراج ، ولكن
مع القول بأن الوسائل التي تعرف شكل مجموعة من البرامج الثانوية المكتوبة باللغة
نفسها . وهناك ثلاثة مفاهيم ممكنة :

- تعريف توسيعات اللغة ذات الاستعمال العام ، للسماح في لغة آدا ، بإجراءات إدخال -

إخراج لا تفرض مشاكل العمل والإستعمال التي أثارناها ، والاستفادة من مجموعة اللغة وتوسيعاتها ، يجب أيضاً فرض أكبر قسم ممكن من جسم الرزم المناسبة للعاملين .

- تعريف توسيعات باستعمال عام ، كما هو أعلاه ، ولكن دون البحث عن كتابة الرزم المناسبة في آدا ، أجسام الرزم المعروضة ستكون غير كاملة ، وليس لها أية قيمة وصفية .

- تعريف التوسيعات الخاصة ، بإضافة إنشاءات جديدة للأنواع وبيئات جديدة للدخال والإخراج إلى اللغة ، وغير صالحة للاستعمال خارجها .

13.4.3 حساب ختامي

طول هذا الفصل يبدو وكأنه مفاجئ ، لأنه يعالج مشاكل صعبة . ويدور لنا أنه القسم الأكثر أهمية للاختبار عندما نقيّم اللغة ، لأنه يُشكل تحليلاً لامكانياتها ، وانتقاداً لها .

إذا كانت اللغة تقدّم وسائل خاصة للإدخال-الإخراج ، فهذا محتمل لأنها لا يمكن أن تقوم بخلاف ذلك : لن تكون قوية لجهة التقطيع الزجلي ، الشمولية ، والقدرة على التوسيع .

ولو كان العكس ، نقوم بتعريف مسبق لأولية الإدخال-الإخراج في إنشاءات من اللغة ، كما في الغول 68 ، في لغات كتابة الأنظمة وفي آدا ، فهذا لأن اللغة تمتاز بإمكانيات حقيقية .

الخدمات المقدمة بواسطة مختلف الرزم ليست أبداً سهلة للاستعمال . كثير من الأوامر هو فائض ، ولا يُستخدم إلا لتقنيع عدم الكفاية في اللغة . فعالية اللغة تبدو مشكوكاً بها ، فيما لو جرت معالجة الرزم المحددة مسبقاً بطريقة عادية ، وهذا هو السبب الرئيسي لتفاذي إدخال إنشاءات خاصة إلى اللغة . الإمكانيات الحالية هي غير كافية بالنسبة لمشاكل الإدارة ، وبالنسبة لبعض النقاط ، وفي المقابل ، فهذه الامكانيات هي أرفع مستوى من الإمكانيات الموجودة في لغات كتابة الأنظمة (I.S, MODULA...) ، وفي حقول خاصة للتطبيق ، الامكانيات المقدمة في لغة آدا يمكن أن تعتبر وكأنها مقبولة تقريباً .

الفصل الرابع عشر

العناصر النحوية ، اللغوية والنصية

راجعته :

هذا الفصل الذي يأتي في الأخير ، لا يتبع بنية الفصول السابقة ، لأنه لا يتناسب لا مع مفهوم آدا ولا مع فصل محدد من [MR] . في قسمه الأول ، سنحاول تحليل شكل أوصاف اللغة ، وبعد ذلك الملاحظات الخاصة التي تمت في كل فصل ، وسنقوم بتحليل الشكل النحوي . أما نهاية القسم الثالث من هذا الفصل فهو مخصص للمفاهيم اللغوية .

14.1 أشكال أوصاف اللغة آدا

14.1.1 الشكلين للوصف

عرُفَت آدا بطريقتين :

أ - كما بالنسبة للغات الكلاسيكية ، فالمرجع المساعد [MR] يعرف آدا جزئياً بواسطة نحو شكلي ، وينص باللغة الإنكليزية .

ب - في الإتجاه المحدث ببعض المحاولات (تعريف متساوي لـ $PL/1$ ، تعريف من بروكسل . . .) ، فإن آدا هي موضوع تعريف إلزامي .

14.1.2 الوصف النحوي

هذه اللغة هي معرّفة جزئياً بواسطة تشكيل من قواعد - C . يقلّم هذا التعريف بطريقة قريبة من التعريف Backus-Naur .

من الممكن إجراء بعض الملاحظات على هذا الوصف الشكلي :

أ - أبجدية نهائية

ليس معرّفاً بشكل إلزامي . النحو التافهني يقف في حدود « التعابير النهائية »
التالية :

character_literal
digit
lower_case_letter
other_special_character
space_character
special_character
underline
upper_case_letter

يجب البحث عن تعريف لها (غير إلزامي) في [MRA2] (العناصر اللغوية) .

ب - بدئية

لا يوجد أبداً بدئيات في آدا . ليس كما جرت العادة فمفهوم « البرنامج » هو غير موجود . البدئية هي بلون شك مفهوم « التصريف » .

ج - الحشو

الأربعة مفاهيم

compilation exponentiating_operator
logical_operator pragma

هي معرفة ولكنها لا تُستعمل أبداً (من الصحيح أن مفهوم التصريف يجب أن يستعمل كبديئية) . ومن وجهة نظر شكلية ، المفاهيم الثلاثة الأخرى المذكورة هي حشو .

أما الـ 11 مفهوم

base integer_type_definition real_type_definition
exponentiating_operator null_statement condition
loop_parameter operator_symbol formal_parameter
actual_parameter task_declaration

هي وحسب تعريفها المتشابه مع مفهوم آخر ، تؤلف إستعمال مزدوج مع هذه الأخيرة .

د - النقص

النحو المقدم لا يمكن أن يُعرّف سوى لغة C- داخلية في اللغة آدا . هذا النحو قد يطمح لتعريف الأصغر من هذه اللغات C- ؛ ولكنها لا تقوم به . وبشكل آخر ، فإن بعض عمليات التنفيذ المقدمة بطريقة غير إلزامية قد تصبح شكلية دون الخروج من الإطار النحوي المعتمد . اللغة C- التي نحصل عليها ستكون قريبة من اللغة آدا الحقيقية . من

الأمثلة العديدة على التخصيص المطلوب سنه ، فنشير إلى نداءات المبرمج الثانوية [MR 6.4] بدلاً من القول :

« إن المتغيرات المؤشرة والمتغيرات المسماة يمكن أن تتم في نفس النداء ، مع المحافظة على كون المتغيرات المؤشرة تأتي في رأس الموقع الطبيعي ، أي ، بعد متغيرة مسماة لا يمكن لباقي النداء أن يحتوي سوى على متغيرات مسماة » .
إذاً من الممكن تعريف التعبير « قسم - متغير فعلي » .

```
actual_parameter_part ::= (actual_parameter
                           | actual_parameter | parameter_association | )
                           | (parameter_association | parameter_association | )
```

هـ تقديم

يُقدّم النحو بشكل قريب من Backus-Naur .

وهو يُستعمل [] بالنسبة للصيغ المعتمدة و { } للتعاريف بدلاً من التعريف المتتالي ، مما يجعل التعريف أكثر وضوحاً ومقروءاً . مثلاً .

- استعمال الخط في الفراغات (Underline) في التعابير المتعددة بين عدة كلمات يجعل القراءة مريحة . مثلاً :

lower-case-letter

يحتوي النحو أيضاً على شكل ملاحظات معتمد يحمل تأشيريات دلالية . من الناحية الشكلية فهو لن يغير شيئاً . هذه المؤشرات تكتب على الشكل التالي . مثلاً :

boolean-expression

هكذا ، فهذه المؤشرات تؤلف إطلافاً لتطوير القواعد C- نحو قواعد بمستويين .

هكذا فمفهوم المثال أعلاه يثير قهراً المفهوم الشديد في القول 68 :

proposition ... de mode MODE avec MODI = boolean

مما يجعلنا نتأسف لأن تطوير هذا النحو قد بقي في نقطة الانطلاق .

هذا النحو يعرف 156 تعبيراً (« أوفئة نحوية » ، أو عنصر من ألفباء غير نهائي) . بواسطة نفس العدد من قواعد الانتاج المتعددة . هذا ما يجعل اللغة « كبيرة » (أي أكثر تعقيداً من القول 68 أو باسكال ولكنها مفهومة بشكل أكثر من Algol 68 ، P1.1 أو 1.1S . هذه القواعد هي مذكورة في [MR] . وجرت مراجعتها في نهايته . هذه المراجعة تفتقد لكل مساعدة عند الاستشارة (ترقيم القواعد ، عودة الى

التعاريف وعمليات الاستعمال) . ولكن هذا العمل قد تم بواسطة المستعملين ،
ويبدو أنه قد صُنِعَ في [MRA] .

● وفي النهاية ، نشير إلى أن القواعد النحوية هي مرقّمة كما هو مطلوب في
البرنامج . هكذا فالتعليمة if هي معرفّة كما يلي بواسطة :

```
if_statement :: =
  if condition then
    sequence_of_statements
  } elsif condition then
    sequence_of_statements {
  [ else
    sequence_of_statements ]
end if ;
```

من هنا الطلب في تطبيق if, else, end if ، كما يلي :

```
if condition then
  instruction 1 ;
elsif condition then
  instruction 2 ;
elsif condition then
  instruction 3 ;
else
  instruction 4 ;
end if .
```

هذه الطلبات هي عبارة عن جهد مهم لنموذجة الفقرة ، ولكن وللأسف ، فهي
تحتوي على تناقضات مع الأمثلة (loop statement مثلا) وهي غير كافية للإجراء
الفعلي لل فقرات .

14.1.3 القسم غير الشكلي .

بصفتها غير الشكلية ، فإن القيود الداخلة الى اللغة المعرّفة شكلها تحدّد بأن تكون
غير كاملة (منسية) أو كاملة بشكل كبير (تناقض) . نفس الشيء بالنسبة للدلالة .
سنعطي أدناه أربعة أمثلة ، مستخرجة من الفصول الأكثر كلاسيكية من [MR] .
هذه السيئات يمكن أن تُصَحَّح في [MRA] ، لهذه الأمثلة الدقيقة ، ولكن يبقى هناك
أخرى ، مخبأة ، في الأقسام الرئيسية للغة .

مثال رقم 1

يمكن أن نكتب :

3 # 2 أي 3 بقاعدة 2 ؟

النحو التشكيلي كان يسمح به ، لا يوجد قيود غير شكلية [MR 2.4.1] تمنعه ؛
والقارئ وحده سيشرح بعمق أن هذا لم يكن يناسب أبداً تعريف عدد بقاعدة 2 . هذه
الكتابة كانت ممنوعة بإضافة قاعدة غير شكلية [MRA 2.4.1]

المثال رقم 2

الأعداد هل هي مواضيع ؟ [MR 3.1] يبدو وكأنه يقول لا ؛ التصريح عن
الموضوع (معطيات مثلاً ، في القاعدة [صفحة MR 3.2]) .

LIMITE: constant INTEGER : = 10-000; عدد ثابت

كان يؤدي إلى الاعتقاد بأن الجواب هو نعم ؛ والتصريح

LIMITE: constant : 10-000; عدد -

كان شبيهاً بالتصريح عن موضوع . العدد هو فعلياً عبارة عن موضوع من نوع
« صحيح غموضي » . [MRA] - يجب أن يعرف الأنواع العامة النموذجية والعمليات
المعتمدة .

مثال رقم 3 :

[MR صفحة 6.2 في الأعلى] : معرف البرنامج الثانوي هو قبل كل شيء داخل ،
ويمكن أن يستعمل لاحقاً . يجب أن نقدر على كتابة :

- ممنوع function F (X : INTEGER : = F(0)) return INTEGER

ولكن هذه القاعدة هي متناقضة مع الفقرات السابقة (جسم الدالة يجب أن يكون
مصمماً) . مع البدء برؤية المعرف F بعد مواصفة البرنامج - الثانوي ، [MRA]
تجعل هذه الكتابة غير مسموح بها .

مثال رقم 4

في [MR 8.1] ، يُقال إن نطاق النص أو التصريح له تأثير يبدأ في النقطة التي يجري
فيها إدخال المعرف . ولكن يُقال أيضاً : [MR صفحة 8.4 في الأعلى] إن المعرف
المحبا (بنتيجة تصريح مجانس لإنشاء داخلي) هو غمباً أيضاً في كل الإنشاء الداخلي .

أما التصريح الثاني فيقوم بعمل تجبئة شيء ما قبل النقطة التي تظهر فيها . وهذا هو
تناقض مع التأكيد الأول . ولحسن الحظ ، فإن التأكيد الثاني هو خطأ [MR] ،
والتناقض يسقط !

بدون شك فإن بعض القراء يمكنهم أن يكتشفوا ما ينقص هذه اللغة في القسم غير

الشكلي من [MR] ومختلاً في [MRA] . ومن الممكن أن يكون نفس القراء موافقين على ما يجب أن يكشف . ولكن المرجح المساعد ليس لعبة اكتشاف ، مهما يكن مهما .

14.1.4 وحدات محدّدة مسبقاً

إضافة إلى اللغة ، فإن تعريف لغة آدا يحتوي على بعض الوحدات المحدّدة مسبقاً والموصوفة في الملاحق [MR, A, B, C] ، وهي :

- الخاصيات المحدّدة مسبقاً (مثلاً : FIRST, VALUE, ADDRESS) الموصوفة في النحو بواسطة

attribut : = name' identifier

إذا ما نقوم به هو « المعروف » الذي هو فعلاً الخاصية .

- اللوائح (pragmas) المحددة (مثلاً : IN-LINE, INTERFACE, PACK) المسبوقة نحويّاً بواسطة الكلمة المحفوظة pragma ؛

- الرزمة STANDARD التي تقوم بإدخال :

- أنواع وأنواع - ثنائية محدّدة مسبقاً (مثلاً : SYSTEM , REAL , INTEGER , NAME) .

- السمات المحدّدة مسبقاً .

- العمليات المحدّدة مسبقاً .

- الاستثناءات المحددة مسبقاً .

- المكتبات المحددة مسبقاً وبشكل خاص الرزمة الضرورية TEXT-IO .

هذه الوحدات المحدّدة بشكل مسبق هي قسم من تعريف اللغة (أي عاملة على كل مصرف) وتزيد من التعقيد الحقيقي للغة ، وتعليمها وإستعمالها ، ولكنها تسمح بفصل المفاهيم المتعلقة بالمكنة عن اللغة .

14.1.5 البرنامج المزيف

إضافة إلى اللغة (« برامج صحيحة ») هناك مسألة تختلف البرامج « المزيفة » ، التي تبدو وكأنها تُشكّل غيباً على لغة آدا الصحيحة . وهي تحتوي على نوع من الدلالة ، لأن عملها محدّد [MR 1.6] : مثلاً ، إعطاء نصوص للأغلاط . هناك أربع فئات من البرامج المزيفة ، وهي :

أ - البرامج التي تُخالِف قاعدة اللغة والتي يجب أن تُعرّف وكأنها مزيفة عند التصريف .

ب - البرامج التي تقدّم حالات إستثنائية : الخطأ هو قابل للإكتشاف عند التنفيذ . وفي

بعض الحالات ، يتبَّه المصْرُفُ « الجَيْد » منذ التصريف ، من أنَّ حالة استثنائية معيّنة قد لا تخفّق في الوصول إلى مرحلة التنفيذ .

ج- البرامج المغلوطة ، يتعلّق ذلك ، وبالأخص بالبرامج ذات التأثيرات الجانبية ، والتي يكون تصميمها غير معروف بالكامل [MR] (وهي تناسب عادة عمليات التصميم المتّحدة في الفول 68) . مثلاً : مفهوم دالة الإدخال DATA ، البرنامج الذي يحتوي على

DATA / DATA

هو مغلوط . ولكن البرنامج هو غير مغلوط ، إذا كان تبادلياً عند الضرب ، وإذا كان يحتوي على

DATA * DATA

فالتعبير الأول المذكور هو غير مغلوط في الحالة التي تكون فيها المعطيات (المطلوب قراءتها في أي ترتيب) متعادلة ، أو معكوسة ، والنتيجة هي إذاً مستقلة عن ترتيب المتأثرات ، أو إذا كان هناك تحميل زائد من نوع « / » .

ملاحظة أن البرنامج هو مغلوط بالنسبة لتعريف آدا ، ولن يكون بشكل عام مُزيّفاً بالنسبة لعمل معين ، والذي يرفع بالضرورة ويشكل أو يأخر أسماء اللا - تعريفات المتروكة في اللغة . هكذا فالمبرمج الكاتب لبرنامج مغلوط لا يمكن أن يدعي أنه قابل للنقل ؛ عملية إكتشاف إن البرنامج هو مغلوط ، لا يمكن أن تتم بواسطة أداة : يتعلّق ذلك بقاعدة تُعطى للمبرمج .

د - البرنامج غير المسموح به [MR 10.5] هو برنامج بدون أي ترتيب في تصميمه . هكذا برنامج لا يمكن أن يتم تنفيذه .

وللإنهاء ، البرنامج الذي « يُزرع » للتنفيذ ، دون ذكر شيء عند تصريفه ودون التعرف على حالة إستثنائية ، لا ينتمي إلى أية فئة معتمدة . وهذا لا يمكن أن يتم . وإذا إستطاع العاملون تأمين هذا الطلب ، فهذا سيُشكّل إختباراً جيداً لنوعية لغة آدا .

4.1.6 التعريف الشكلي للغة آدا

قام بهذا التعريف في INRIA ، G. Kahn ، V. DONZEAU-GOUGE ، B. Lang بالتعاون مع B. Krieg-Brueckner و J.C. Héliard الذين يشكلون قسماً من الفريق الذي عرّف لغة آدا . وجرى نشره في [DF] ومواضيعه هي :

- تعريف دقيق للغة .

- المساعدة في تصوّر اللغة وفي تنقيح المساعد .

- تعريف التمثيل النموذجي لبرنامج آدا .

النقطة الأولى هي موجهة إلى اختبار خصائص اللغة والبرامج ، ومساعدة العاملين الجدد والمبرمجين .

لن يتم دراسة هذه الوثيقة في إطار هذا الكتاب . فهي تبدو وللوهلة الأولى أنها غير مبلوغة بالنسبة « للمبرمج - الوسط » .

14.1.7 الصلاحية

بالتوازي مع هذه الأعمال في تعريف اللغة ، هناك مساعد العامل [Gil] ، الذي يشرح اختبارات المصِّرف آدا ، في الإطار العام للمشروع DOD . يجب أن نتفق بأن لا تصبح الدلالة في لغة آدا معرَّفة ضمناً بالتوافق مع الاختبارات الرسمية .

ومن المثير في هذه الوثيقة ، وعلى الأقل في صيغتها الأولى هو وجود نقاط قليلة الواضحة إما ظاهرياً وإما ضمناً .

14.1.8 ثبات التعريف

ثبات تعريف لغة الغول تم خلال 10 سنوات ، أما تعاريف باسكال فلا تزال عرضة لعدة تعديلات غير متوافقة . لذلك قررت DOD عدم الوقوع في فخ الصيغة المتتالية . ولكن وبعد سنتين من تعريف اللغة [MR] ، ظهر المعيار ANSI [MRA] . وهذا المعيار يختلف عن [MR] بتصحيح مختلف الأخطاء ، وبتنقيح جعلها أكثر قابلية للفهم . ولكن العاملين الذي بدأوا بالعمل باللغة قد اكتشفوا بعض المشاكل ، وعندما يبدأ المستعملون باستعمالها ، فهل سيكتشفون نفس الشيء ؟

14.2 النحو - مفاهيم نصية

المعايير الرئيسية للحكم على نحو اللغة (من وجهة نظر المستعمل) هي ، من جهة ، السهولة في الكتابة والتعلم ، ومن جهة أخرى سهولة القراءة ، ومع هذين الاتجاهين يتبع :

- وضوح في بنية تنظيم التطبيق ، أو البرنامج أو قسم من البرنامج . هذه السهولة في القراءة تضاف إلى غمط اللغة .

- الفهم السريع للبرنامج : يجب أن يسمح النحو بمساعدة الدلالة في البرنامج . في الفصول السابقة قمنا باختبار الشكل النحوي للمفهوم الدلالي . سنحاول إجراء تحليل للملاحظات على النحو وعلى غمط اللغة آدا .

14.2.1 النقاط الإيجابية

14.2.1.1 تنظيم البرامج

وجود تصريف فعلي منفصل ، والفصل النصي بين قسم مواصفة الوحدة وجسم هذه

الوحدة يُسهّل الكتابة (فصل بين مواصفة خارجية والتنفيذ) ، التوثيق إضافة إلى فهم تنظيم البرنامج .

هكذا ، فتعليم هذا التنظيم في آدا ليس فقط سهولة في نحو اللغة التي لا تعرف أبداً مفهوم البرنامج . هذا المفهوم جرى رفضه بسبب منهجية البرمجة .

14.2.1.2 أهلة الانشاءات

تُشكّل البنى (التعليمات المركبة والأنواع) من أهلة بشكل متجانس وبسيط . أمثلة :

```
if ... end if
case ... end case
loop ... end loop
record ... end record
select ... end select
```

إستثناء في أغلب الحالات : end تغلق معالجة الإستثناء .

```
exception exception_handler | exception_handler |
```

هذا التجانس موجود أيضاً في محاكاة كتابة التعليمات case والفقرات بحقول متحولة ، ويجب أن تسهّل عمل المبتدئين . البنى المجهزة بأسماء يجب أن تنتهي بشكل عام بواسطة end متبوع بهذا الاسم ، مما يؤدي إلى الأهلة :

```
procedure NOM is ..... end |NOM| ;
package NOM is ..... end |NOM| ;
task body NOM is ..... end |NOM| ;
accept NOM do ..... end |NOM| ; -- pourquoi do ?
NOM : ..... end |NOM| ;
```

14.2.1.3 النقطة - الفاصلة

النقطة - الفاصلة ليست عنصراً لغوياً خاصاً . ولكننا نعرف إن هذا الرمز هو مرتبط عادة بنهاية السطر وهو سبب أخطاء شائعة ، صعوبات في التعليم ، وهو يؤلف رمزاً يستخدم كنقطة ترميم الخطأ بالنسبة للمحلّل النحوي .

في لغة آدا ، النقطة - الفاصلة لا تُشكّل فاصلاً للتعليمات ، ولكن أداة لإنهاء : تنتهي كل تعليمة بنقطة فاصلة ، مستقلة عن النص . هذه الملاحظة هي مهمة لأن التعليمات البسيطة ، التي تأخذ سطرًا واحداً ، ستكون مكتوبة بواسطة نقطة - فاصلة في

نهاية السطر ، ويمكن تحريكها وإلغاؤها دون تعديل التعليمات . وهذا هو تحسين (بالنسبة للكتابة) بالنسبة للغة ألغول 60 وباسكال حيث النقطة الفاصلة تتعلق بالنص وحيث تتعقد المسألة مع التعليمات الفارغة والتي تحتوي على أهلة (if مثلاً) . وتشكل النقطة - الفاصلة فاصلاً في التصريحات .

وفي آدا ، تعتبر النقطة - الفاصلة كفاصل في لوائح المتغيرات الشكلية في برنامج ثانوي . . عدم التجانس بين التعريف الشكلي للمتغيرات وكتابتها عند النداء هو منبع شائع للأغلاط .

14.2.1.4 الفراغ

يجب أن نوضح التعليمات الفارغة ، أو اللاتحة بالمركبات « فراغ » (في تسجيلية record) ، بواسطة الكلمة المحجوزة null (متبوعة بنقطة فاصلة) . ولكن ، الكلمة null هي قيمة يمكن أن تُخصَّص إلى موضوع من نوع access والذي لا يعني أي موضوع . ولأن آدا هي لغة حيث مفهوم النوع هو أساسي ، وحيث يدوم التفريق expression-instruction ، فمن المؤسف استعمال نفس الكلمة المحجوزة لتحديد ثلاث وحدات من طبيعة مختلفة تماماً . في أغلب الحالات لا يوجد ، هنا ، مشكلة في إمكانية القراءة (مفهوم الاستعمال هو بديهي) ، شرح الفراغ هو عامل تحسين لهذه القراءة .

14.2.2 النقاط السلبية

14.2.2.1 الكلمات المحجوزة والرموز

يوجد في لغة آدا 62 كلمة محجوزة ، أكثر من رمزين مركبين يلعبان نفس الدور (N و <>) ، هناك 37 في باسكال و75 في اللغة LIS . يجب الإشارة إلى أن بعض الكلمات التي يمكن أن تكون محجوزة هي مستبدلة بواسطة تعابير محدَّدة في الرزمة النموذجية (مثلاً WORD, BIT) . ومع أن عدد الكلمات المحجوزة هو كبير ، يبدو أنه يجب الحصول أيضاً على فائدة لتفادي الاستعمال الزائد لبعض هذه الكلمات . بعض عمليات الاستعمال لها مفهوم طبيعي خارج - النص ، ولكن يمكن أن تؤدي إلى برامج غير مقروءة أو غير مفهومة .

الجدول 1 يعطي بعض الأمثلة على إستعمالها الزائد .

14.2.2.2 الإنشاءات النحوية

بعض الإنشاءات هي ثقيلة . مثلاً :

type VECTOR is array (INTEGER range <>) of BOOLEAN ; -- ouf !

هذا النحو هو ، أكثر طبيعية لأنه يعطي الانطباع بأنه يتعلق بجدول ديناميكي ، بينما

هو يتعلق بنموذج جدول (لا يمكن تعريف أنواع من هذا النوع) . سيكون من الأفضل إدخال كلمة محجوزة أخرى ، مثلاً :

pattern VECTOR is array (INTEGER) of BOOLEAN ;

● ب - الوصف النحوي هو غالباً قليل الدقة وبعض الإنشاءات هي غير إسقاطية .
مثلاً (التصريح عن الموضوع) .

object_declaration :: =
 identifier_list : [constant] subtype_indication [:= expression] ;
 | identifier_list : [constant] constrained_array_definition [:= expression]

كما هو مكتوب ، هذا الأمر لا يؤدي إلى الإلزام بإعدادات وتهيئة الموضوع الثابت .
إضافة لذلك ، فمن الممكن إدخال نوع مجهول (وليس فقط نوعاً ثانوياً) مما يمنع لجميع الأنواع الأخرى ، ما عدا بالنسبة للمهام ، وهذا ليس إسقاطياً مثلاً .

COLOR-TABLE: array (1..N) of COLOR;

● النحو يؤدي غالباً إلى عدة أشكال متساوية . هكذا فهل يسمح هذا النحو بالكتابتين 1 و2 لتحديد شرط المميز

1) LARGE: BUFFER (200);

2) MESSAGE : BUFFER (size N 200);

يمكن أن يُحتفظ بنحو واحد ، برأينا فهو (2) ، لأن الأول لا يعني شيئاً ، بالرغم من الاستعمال الداخلي للرمز => في عمليات التصغير والاعداد .

كلمة محجوزة أو رمز	إستعمال رقم 1	إستعمال رقم 2	إستعمال رقم 3
with	مواصفة نص (with)	متغير شكلي أصلي من نوع برنامج ثانوي	
or	مؤثر منطقي (تعبير)	court-circuit دائرة مفتوحة or else	تعلیمة select
else	تعلیمة if	دائرة مفتوحة (or else)	تعلیمة select
then	تعلیمة if	دائرة مفتوحة and then	
when	تعلیمة case record	إستثناء	تعلیمة select
null	تعلیمة فارغة	فقرة فارغة	متحولة بلوغ لا تدل على شيء
mod	مؤثر ثنائي (تعبير)	تسطير at mod	
in	طبيعة المتغير	إنتهاء إلى range	إعادة تسمية في generic
for	حلقة	مواصفة عمل	
use	use أمر	مواصفة عمل	
new	نوع مشتق (تعريف)	تخصيص (إنشاء موضح)	إنشاء نموذج
select	جهة مناوبة : إنتظار جدول	جهة مناداة : إنتظار إنتقالي	
	range (جدول غير إلزامي)	مختلف الأنواع الشاملة	
	شرط when	تناسب بين المتغيرات الشكلية ومتغير فعلي	3 (مجاميع 4) شرط المميز

علاج	نتائج
كلمة أخرى محجوزة ، أو أعلّمة حول كامل قسم المتغيرات الشكلية الشمولية .	إستعمال 2 : أقل طبيعية من الممكن دلالة غير بديهية
1 - and then or else ستكون مؤثرات معينة (غير الكلمات المحجوزة أو الرموز المركبة)	إستعمال 2 : then , else , or يؤدي إلى if غير مقروءة
2) نحو select للمراجعة	إستعمال 2 : غير متجانس
جعل النحو أكثر تجانسا	الاستعمالان 2 و 1 ليسا متجانسين مع case
بلوغ إلى nil = nul	كلمة تعني الوحدات المختلفة
all هي ليست طبيعية بقدر ما يبدو يجب أن نفهمها كعنوان mod للتعبير	
كلمات أخرى لاعادة التسمية	
غير مستعملة	حل زائد ولكن لا يوجد مشاكل قراءة
محجوزة مختلفة	لا تساعد في فهم الدلالة
نحو اخر لجدول بدون شروط إلزام	مفهوم طبيعي بتعبير مهمل . مشكلة في إمكانية القراءة
إقامة تناسب بين المتغيرات باستعمال : : :	رمز غير مريح

جدول 1 - حول إستعمال الكلمات المحجوزة والرموز

● د- النحو هو غامض ، مما يؤدي إلى صعوبة في فهم البرامج (إيهام عند القراءة) . مثلاً : (مخصّصات ، [MR 4.8] .

```
allocator :: = new type_mark [ (expression) ]
              | new type_mark aggregate
              | new type_mark discriminant_constraint
              | new type_mark index_constraint
```

الجملة التالية :

```
new type_mark (expression)
```

تنتمي إلى الإمكانات الثلاث الأولى و new type-mark (ident) إلى المجموعة .
(تقييد دلالي [MR 4.3] يمنع هذا الشكل الخاص من الاستعمال للمجاميع) . وجرى
تصحيح هذا في [MRA] ، ولكن هل أصبح خالياً من الأخطاء الشبيهة في داخله ؟

● ه- بعض التعابير هي سيئة الاختيار ، لأنها قريبة ، وبدون إدخال ضبابية في
القواعد ، تجعل إمكانية القراءة صعبة . مثلاً : استعمال نوع النتيجة كأسم تحويل [MR
4.6] ، وبالأخص في حالة الأنواع غير المحددة :

```
type SUITE is array (INTEGER range <>) of INTEGER ;
GRAND LIVRE : array (1..100) of INTEGER ;
A : SUITE (1..100) ;
INDICE : INTEGER range 1..100 ;
B : INTEGER ;
A := SUITE (GRAND LIVRE) ;
B := GRAND LIVRE (INDICE) ;
```

- تخصيص لجدولين بعد التحويل ؛
- عنصر جدول

● و- يشكل عدم التجانس منبهاً لبعض الأخطاء المحتملة والابهام . هكذا فالأمر
End يجب ، حسب الحالة ، أن يكون متبوعاً بلا شيء ، بكلمة - مفتاح أو بمعرف .

```
end ;
end if ;
end MON_BLOC ;
```

نفس الشيء ، إذا كان يوجد tasse body و package body ، ولا نجد لا function
body ، ولا procedure body .

● و- بعض الأشكال تختلف عن العادات :

- وسم
- الجملة for في حلقة .

14.3 مفاهيم نحوية

في وصف اللغات ، لا تشكل العناصر اللغوية شكل عام الموضوع إلا في عدة أسطر (هكذا في آدا : ME1.2 ، MR2) . وهي لا تقدّم أية صعوبة نظرية ، وتشكل موضوع قليل من الملاحظات ، كما في التعريف كما في الأعمال التربوية . إحدى النقاط الأكثر إزعاجاً أن نرى عادة وبشكل مكرر ، على طريقة « إستقلالية بالنسبة للمكنة » ، الشكل الفيزيائي للبرامج قليل الأهمية .

العناصر اللغوية هي موضوع [ME1.2 و MR 2.3] . وتؤلف مجموعة السمات ، والملاحظات .

14.3.1 مجموعة السمات

وهي معروفة وكأنها تتألف من :

● مجموعة أساسية من 37 سمة تسمح بكتابة كل برنامج :

.. 26 حرفاً كبيراً لاتينياً .

.. 10 أرقام

.. فراغ

.. 20 سمة خاصة التالية :

! _ < = > ; / . - , + * () ' % & # " ~

● من 38 سمة إضافية غير إلزامية :

.. 26 حرفاً لاتينياً صغيراً

.. 12 سمة خاصة أخرى هي : ! ~ | { } ^ _ [\] ' " & # \$ %

المجموعة من 57 + 38 سمة تناسب 95 سمة مطبوعة للكمبيوتر ASCII . السمات 12 الخاصة الإضافية لا يمكن أن تستعمل إلا في سلاسل السمات ؛ وخارج هذه الأخيرة ، الأحرف الصغيرة هي متساوية بالكبيرة المناسبة . [MR] تستعمل هذه الأحرف الصغيرة للإشارة إلى الكلمات المحجوزة مثل begin . هذه 38 سمة لا يمكن أن تستعمل لتعريف المؤثرات الجديلة .

كل سلسلة مكتوبة في المجموعة الكاملة يمكن أن تكتب أيضاً في المجموعة الدنيا باستعمال معرفات محدّدة في الرزمة ASCII [MR ملحق C مؤثر الضم (المؤثر &) .

مثال :

- في المجموعة من 95 سمة « QUI ETAIT ADA ? »

- في المجموعة من 57 سمة ASCII . Query & « QUI ETAIT ADA »

هذه السمة & تناسب مؤثراً معيّنًا ؛ وليس لها سوى دور نحوي . نعتقد أنه كان من

الأفضل السماح بإهمال هذا المؤثر (ضم ضمني في I.IS) وعدم وضعه إلا ليدل كاتبه الفقرات أو المصنفين مثلاً أن تحضير البرنامج لن يتم تهديده .

إلى هذه المجموعة من ٩5 سمة « مرئية » ، يمكن إضافة (واستعمال في السلاسل) جميع السمات ASCII المستعملة بواسطة أنظمة الإرسال والمُدعوة غالباً سمات تحكم .
فلنشر مثلاً إلى :

« للجدولة الأفقية » ASCII.HT
« لعودة المنزلة » ASCII.CR

كما يسمع بكتابة ، وبشكل كامل في آدا ، برامج دون الاهتمام بما يوازيها في التعبير الرقمي لكل ما يمكن قراءته ، مثلاً ACK أو NAK . ولكن تأثيرها يتعلق باستعمالها [MRA 2.2] .

تقييم

- وجود المجموعة الدنيا هو مهم لأنها تسمح باستعمال « العتاد القديم » للمكنات ، كالمقاييس IBM 029 أو الحاسبات cyber من شركة Control Data التي تمتاز بمجموعة مؤلفة من ٩3 سمة فقط .

- التكييف يتم بشكل موفق في الأعلى : كل برنامج مكتوب بالمجموعة من 57 سمة يمكن أن يدور هناك حيث مجموعة السمات ٩5 هي مقبولة ، ولكن ، وهذا هو المهم ، كل برنامج مكتوب بالمجموعة من ٩5 سمة يمكن أن تعاد كتابته بسهولة كي يعمل على مكنة تستعمل فقط 57 سمة مختلفة ، « بسهولة » لأن ذلك ليس معقداً للكتابة . ولكن البرنامج الناتج لا يمكن دائماً أن يكون مقروءاً . فلنقارن مثلاً الأسطر التالية :

a) WRITE ("a [i] ");
b) WRITE (ASCII.A & ASCII_BRACKET & ASCII.L
& ASCII_R_BRACKET);

- يوجد في بعض الأحيان تبعية بالنسبة للعتاد الذي وُضِعَ بصمته لمعرفة ما يجب طبعه على « مكنة آدا » عندما نستعمل المجموعة من ٩5 سمة بينما أعضاء الإخراج لا تتمتع بجميع السمات المناسبة (ماذا يحدث بالنسبة للأوامر WRITE السابقة على طباعة كلاسيكية ؟) .

- العمل الأساسي المطلوب القيام به ، هو إن عمليات الإستعمال الوطنية ليست معتمدة : ASCII تعني American Standard وهذا فإن الألفباء الوحيدة المستعملة هي اللاتينية : لا يوجد أحرف مع accent ، ولا إشارات متعرجة . لا يمكننا طباعة النصوص التالية [MRA 2.6] .

Français : Là où Moïse aperçut le dôme ...
 Espagnol : ¿ Donde está tu niño ?
 Croate : Đorđe Džadžić

ولكن [MRA] ليس واضحاً . لا نعرف أبداً ما إذا كان باستطاعتنا الحصول على الرزمة الخاصة به . ولا ما إذا كان يجب أن تتناسب السمة « [مع المعروف ASCII-L-BARCKLIT كما هو الحال في الكود ASCII من الكود ISO أو إذا كان من الممكن أن يناسبه «ن» كما في الكود AFNOR من الكود ISO .

- نראה بعض التعابير ليست دائماً سهلة (حتى بالنسبة للمحللين اللغويين سيكون ذلك صعباً) . مثلاً (مستخرج من GI 2.2) .

```
if('la'a..'')then
  if('')then -- deux caractères ' ' et ' '
CH_HEX('A') -- 'A' qualifié par CH_HEX
TBASEFIRST -- nom d'attribut
```

- وفي النهاية ، وبالأخص ، من المؤسف إنه على عكس ما كان يُحاول بلوغه في Algol 60 والذي أصبح حقيقة في Algol 68 ، فإننا لم نفصل وصف اللغة المرجع عن تلك التمثيلات المتعدية مما قد يؤدي إلى أن تصبح هذه المجموعة من السمات قديمة .

14.2.2 مفهوم « السطر » والاتفاقات حول الفراغ
 مفهوم السطر من البرنامج لا يشكّل قسماً من البرنامج (من النحو) . ولكنه متقارب لأنه يوجد عدة طرق للعبور إلى السطر الجديد [MR 2.2] .

اتفاقات فصل الوحدات اللغوية هي بسيطة ، وكافية ودقيقة ، وبكلمة أخرى ، السطر الجديد ، أو الفراغ . هذا الأخير هو ذو معنى في جميع اللغات الحالية ، ويمكن القول في النهاية أن تقدماً قد تم من خلال لغة فورتران : وقد نتذكر المدفع الذي انفجر في فينيا بسبب خطأ (نقطة بدلاً من الفاصلة) لم يكتشف المبرمج الذي اعتبر أن 1.2 = D041 هي تعليمية تخصيص D041 . الاستعمال الشائع للفراغ بالنسبة ، مثلاً ، لكتابة الأعداد بواسطة قطع من ثلاثة أرقام يمكن أن يُستبدل باستعمال خط أبيض .

كتابة 247-234-1 ليست صعبة وهي كما 1234567 (أو 1.234.567 كما تقوم بعض المدارس الابتدائية بالتعليم) ؛ نفس الشيء بالنسبة للجملة Pomme-de-terre فهي أيضاً مقروءة مثل Pomme de terre .

14.3.3 معرفات ، أعداد وسلاسل
 المعرفات هي كلاسيكية . ولكن لنشر إلى استعمال (الفراغ المخطوط) (-)

والتعادل بين الأحرف الصغيرة والكبيرة . عدد السمات المقبولة للمعرف هو محدود بواسطة حجم الأسطر ، ولكن المعارف مستحفظ بجميع هذه السمات ذات المعنى ؟

لا شيء خاص بالنسبة للأعداد العشرية . وعلى العكس ، فإن الأعداد في قاعدة تختلف عن 10 تبدو وكأنها معرفة بتميز للمحولات اللغوية ليس العادات الإنسانية : يجب كتابة

83 في قاعدة 16 بالشكل المثبت وليس 16 # 83 # et non 83 # 16
1011 في قاعدة 2 بالشكل المثبت وليس 2 # 1011 # et non 1011 # 2

في الشكل الكلاسيكي (« ... ») ، لا يمكن للسلاسل أن تكون ما بين سطرين ، ولكن بإمكاننا ضمها بواسطة المؤثر الفائدة الكبيرة من ذلك هي في حالة خطأ في () غياب ، تثقيب سيء ، الخ) ، يبقى سطر واحد مضطرب لغوياً . وليس كامل نهاية البرنامج بسبب تبديل كل ما هو سلسلة مع كل ما هو ليس بسلسلة .

السلاسل هي قابلة للاتحاد فيما بينها (منذ [MRA]) وحتى بالنسبة للسمات :

"chaîne" & "chaîne"	--	التهجة "chainochaine"
"chaîne" & "c"	--	التهجة "chainec"
"c" & "chaîne"	--	التهجة "cchaîne"
"c" & "c"	--	مسموح cc
"c" & "c"	--	مسموح ولكن يتعلق بالمفهوم

14.3.4 الملاحظات

حتى الآن ، فأي شكل كاف للملاحظات لم يعرض في لغات البرمجة ، جميع مصممي اللغة يرفضون أن يجعلوا موقع الملاحظات نحوياً . تلك المعروضة بواسطة آدا ليست كاملة ، ولكن الحل المعروض بسيط من جهة وفعال من جهة أخرى . إضافة لذلك ، فأي رمز جديد لم يجر إنشاؤه .

فالملاحظات هي كل ما هو موجود بين .. ونهاية السطر . كما في حالة السلاسل ، الخطأ في الرمز - لن يؤدي إلى سطر واحد خطأ وليس عملياً حتى نهاية البرنامج . فلنشر إلى أن 2 .. = X لا يمكن أن تفسر وكأنها (2 -) : X = [MR 4.1] . وهذا يسمح بوضع الملاحظات (بعد التصريحات ، والتعليقات أو بعد then الخ) أو هي على سطر كامل كما في فورتران . سنجد عدة أمثلة في الكتاب الحالي (مثلاً في الفصول 13 و 10) . وبالنسبة للمحلل ، فالملاحظات هي كل ما يتبع الكلمة pragma .

البعض يأسفون لعدم إمكانيات وضع ملاحظات في وسط السطر . وبرأينا ، نحتاج إلى إعطاء قواعد ربط الملاحظات بعناصر لغوية : تحديدات داخلية وشروط الإلزام

(بالنسبة للمبرمج) ستكون معوضة بالاستعمال الأفضل (توثيق ، صيانة ، توليد أوتوماتيكي) لهذه الملاحظات . من المؤسف بأننا لم نتوقع في اللغة أي رمز خاص (مثلاً ++) للملاحظات حيث التصفيح لا يجب أن يلمس بواسطة مقسمي الفقرات أو بواسطة المصنفات ، أو للملاحظات التي ستتحكم بالمتغيرات ، أو أي وسيلة للحصول على مختلف أنواع الملاحظات .

14.3.5 ملاحظات على المحللات اللغوية والدلالية

- من المؤسف أن لا يسمح تعريف اللغة بالحصول على محلل لغوي بسيط . هكذا ففي الحالة التالية

ident (' ' ' ' ' ' ...) ;

- يناسب type-mark

- يأتي من تعبير مُمَيَّز (expression-qualifié)

المحلل اللغوي هو مُلْزَم بالنظر إلى النص المحصور من قبل ، فقط بواسطة نهاية السطر ليعرف أن ' ' ' ليست السمة .

- قواعد اللغة المعنية في [MR] مع أنها تصف لغة كبرى (حتى بالنسبة للمفاهيم اللغوية) هي غامضة بالكامل . فكل عامل يجب أن يعيد كتابة واحدة وإجراء الاختيار في التجميع ، التفريق المحتمل بين هذه الانشاءات اللغوية هو مرفوض في مرحلة الدلالة .

14.4 خاتمة

منذ سنوات ، جرى تقديم كبير في تعريف لغات البرمجة ، واستفادت لغة آدا من هذا التطور .

يوجد في آدا إرادة حتمية كالتي نجدها في LIS بإرضاء المبرمجين الذين يرغبون بمعرفة التفاصيل النحوية . وبالتحديد ، للمبرمج الحق في الخطأ وخط أكبر للحصول في وقت قصير على لائحة بجميع الأخطاء اللغوية والشكلية .

في كل ما يتعلق بالنحو نفسه ، فإن آدا هي لغة بمستوى عال ، حيث جرت المحافظة على الأمور الأساسية فيها . يمكننا للمبرمج بلون أدنى شك بخط اللغة ، ليس بديهياً أن يكون معيار القراءة ، وهو موضوع رقم 1 في دفتر الشروط في DOD ، قد جرى الوصول إليه . فلغة آدا ليست لغة سهلة التعلم ، على الأقل مع المساعد المرجعي الوحيد حالياً .

ولم يعط المؤلفون عناية فائقة بتقديم تعريف نحوي شديد مع أنه كان ذلك من الممكن (قواعد بمستويين ، تعريف شكلي مرثي وكامل) .

الاختصاصيون في اللغة ، والعاملون ، لم يستطيعوا سوى الإصرار كي يتم الانتهاء من هذا العمل . ويفضل الاختصاصيين تم وضع وصف دقيق ومرئي وربما أخطر أكثر سهولة ودقة ، مثلاً ، بواسطة مخططات ، ولكن المحاولات أثبتت بأنه عملياً إذا تكيف هذا الشكل مع الإنشاءات المركبة كتلك الخاصة بلغة باسكال ، فهذا لن يتم دائماً في لغة آدا .

الفصل الخامس عشر

الخاتمة

راجع

هناك معلوماتيون يتابعون ، إما بسبب كونهم محافطين ، وإما بسبب شروط الإلزام ، التراتبية أو الاقتصادية ، كتابة برامجهم بلغة فورتران ، كويول أو أسمبلر ، هؤلاء ، فإن أدا لن تكون لغة بعيدة ، فهي مُزينة بصفات فرضية مجردة وبجميع طروحات البرمجة .

هناك قلة من المعلوماتيين الذين وجدوا في اللغات القوية ولكن المنقودة في بعض الأحيان (Algol 68 مثلاً) إمكانية العمل بكتابة بأسلوب جديد وبرمجة عملياتية وربط التعاريف حسب ترتيب تصوري واختزال عمليات التخصيص الى العدد الأدنى الأساسي والمحد في نص واحد للبرامج القابلة للتنفيذ وللخوارزم القابل للقر . وذلك مع ملاحظة مقدمة التصورات الجديدة الفعالة في لغة تعرف انتشاراً كبيراً ، هؤلاء يأسفون لأن أدا تمنع تطبيق قسم من الأبحاث الجارية خلال السنوات العشر الأخيرة في مادة فن البرمجة .

يوجد في النهاية جيل بثقافة معلوماتية مركزة على :

((COBOL or FORTRAN) or else PLI) and PASCAL;

وهذا الجيل سيجد في أدا التصورات والإنشاءات التي تعود عليها ، يُضاف إليها . . . غنى كبير في اللغة وإمكانات عملية ملائمة للبرمجة النشطة ، التركيبية والزجلية . وتعمل أدا بالتأكيد أفضل من PL/I أو Algol 68 ليس فقط في كتابة « البرامج » أو الخوارزميات ، ولكن المتوجات من البرامج والمناهج ؛ ولا داعي للتذكير بأن لغة أدا هي النتيجة الأولى لمشروع واسع وكبير [Steelman 79] . وحسب نمط هذه اللغة ، فإن أدا تبدو وكأنها لغة برمجة موجهة للاختصاصيين والمحترفين .

(*) ليست « خاتمة » . ليس هذا هو هدفنا ، كما نشرنا في المقدمة ، البحث عن إجماع لرا مجموعة من الناس أو إعطاء رأي نهائي ، جاهدنا لذلك بل محاولة لإخراج نتيجة من خلال الملاحظات المشتركة للمناقشات

الفوائد والسيئات في لغة آدا جرى إختبارها بالتفصيل في كل فصل من الفصول السابقة . بعض هذه السيئات والقيود هو مزيج ولكن عندما يتعلق بالتفصيلات فمن الحكمة أن تمنح أن تؤدي القوة المتحددة للعاملين والمستعملين الى تعديل اللغة ، أو المعيار . وسنحاول أن نوضح هنا هذه الحسنات والسيئات .

الحسنات والفوائد

- المفهوم بعملة مستويات للأنواع ، ثانوية ومشتقة ؛
- نقاوة التركيب البنوي للغة مع ، كنتاج ، إمكانية البرمجة الشديدة والسهولة الصيانة .
- الزجلة والتقسيم ، مفهوم الرزمة وبساطة وسائل التصريف المنفصل .
- وجود أنواع مجردة ومفهوم الأصولية في الإجراءات والرمز .
- وجود تعابير « في الخانات » (تعبير مؤشر) يسمح باجتياز قواعد الرؤيا ، إن بالضرورة أو للأمان .

- كل ما يلمس المهام ، في الوقت الفعلي أو بالتوازي ، والذي لم يكن موجوداً في السابق في كل لغة إنتشار عامة .

السيئات

- عدد كبير من الإستثناءات لقواعد أو لمجموعات القواعد العامة وهذا ما نسميه بالصفة «faible orthogonalité» (إسقاط ضعيف) ؛
- غياب فترات محسوبة ويشكل خاص تعابير شرطية ، إضافة إلى غياب القيم من نوع «procedure» .
- بساطة الإدخال - الإخراج ، وصعوبة البرمجة التي تنتج عنها .

الصلابة في نحو مركبات البرامج : مثلاً ترتيب التصريحات هو إلزامي ولكنه أقل من بامسكال (يمكن أن نأسف إلى محاولة فصل التصريحات عن جسم البرامج الثانوية الذي لم يُدفع إلى النهاية) . هذا الشيء مُضافاً إلى غياب صفة الإسقاط ، قد يؤدي إلى تعقيد كبير في تعلم اللغة وإلى صعوبة في البرمجة مستقبلاً .

جميع هذه القيود وهذا الغياب لها في بعض الأحيان أسبابها ، التي تبدو بالنسبة لأغلب الناس ناتجة عن توازن بين فلسفتين :

أ - فرض مادة وحيدة على الجميع ، أي منع ، في اللغة أو بواسطتها ، كل ما قد يحدث من قبل المبرمجين غير الإختصاصيين من استعمال شيء وغير فعال .

ب - على العكس ، تقديم الحد الأقصى من السهولة الى اللغة مع ترك إمكانية التعليم للمبرمجين بعملة مستويات ، وللمعلوماتيين الفرصة لوضع برامج بمميزات خاصة ،

هذه السيئات تنتج غالبا عن الإختيار الضاغط ، الذي يذهب بدون شك في إتجاه
الفعالية الصناعية ، والتي تلمع بالبساطة ولكننا نخشى ضياع فرصة الحصول على لغة
موحدة تكفي الباحثين والمطوّرين ، وتؤدي إلى إزالة الوهم من تعقيد المعلوماتية .

مراجع

- هذه المراجع هي مقسمة إلى أربعة أقسام :
- الوثائق الرسمية ، المراجعة في النص بواسطة مفتاح بالشكل [XXC.S.P] ، مثلاً [MR 3.4.5] .
 - اللغات المراجعة بأسمائها مثلاً Algol 68
 - الفقرات أو الكتب المذكورة ، والمؤشرة بواسطة مفتاح تحت الشكل [سم سنة] مثلاً [Bouie 80] .
 - بعض المؤتمرات ، المشار إليها بواسطة المفتاح [مدينة سنة] ، مثلاً [Rennex 80] .
 - سنجد في [Zulewski 81 and Wang] مراجع أكثر شمولاً من آدا .
- المراجع الرسمية

REFERENCES « OFFICIELLES »

- [DF] Formal Definition of the ADA Programming Language, Preliminary Version for Public Review, INRIA, Rocquencourt, 1980.
- [GI] ADA Compiler Validation Implementers' Guide, SofTech Inc., Waltham, MA., 1980.
- [ME] Rationale for the Design of the GREEN Programming Language, SIGPLAN Notices, vol.14, n°6, Juin 1979.
- [MR] Reference Manual for the ADA Programming Language, United States Department of Defense, DoD Management Steering Committee for Embedded Computer Resources, Room 3A318, The Pentagon, Washington, DC 20301, 1980.
Voir aussi [Lougard 81], [Kruchten 82] et Lectures Notes in Computer Science n° 106, Springer Verlag 1981.
- [MRA] Reference Manual for the ADA Programming Language, ANSI/MIL-STD-1815 A, Honeywell et Aixys éd., Paris, Janvier 1983.

LANGAGES

Algol 68	[Van Wijngaarden <i>et al.</i> 76], [Afcet 75]
Alphard	[Wulf <i>et al.</i> 76]
CHM	[Branquart <i>et al.</i> 82]
CLU	[Liskov <i>et al.</i> 77]
GREEN	[Ichbiah <i>et al.</i> 79]
LIS	[Ichbiah <i>et al.</i> 76]
Mesa	[Mitchell <i>et al.</i> 78]
Module-1	[Wirth 77]
Module-2	[Wirth 80]
Pascal	[Jensen & Wirth 74]
Russel	[Demers & Donahue 79]

AUTRES REFERENCES

La notation $\leftarrow N$ après chaque référence indique que cet article est cité au chapitre N et celle $\leftarrow \cdot \cdot$ qu'il s'agit de la définition d'un langage cité en divers endroits de cet ouvrage.

Afcet 75

Groupe Algol de l'Afcet. *Manuel du langage algorithmique Algol 68*, Herman éd., Paris 1975.
 $\leftarrow \cdot \cdot$

Banâtre & Ployette 79

J.P. Banâtre & F. Ployette, *Traitement d'exceptions et de fautes résiduelles dans les langages de programmation*, *Bulletin AF'CET-GROPLAN* n°9, *Panorama des langages d'aujourd'hui*, Cargèse, 14-22 Mai 1979. $\leftarrow 9$

Barnes 80

J.C.P. Barnes, *An Overview of Ada, Software - Practice and Experience*, vol.10, n°11, 1980, p. 851-887. $\leftarrow 1$

Bert & Jacquet 78

D. Bert & P. Jacquet, *Some validation problems with parameterized types and generic functions*, *Third International Symposium on Programming (mars 1978)*, Dunod éd., p. 279-282.
 $\leftarrow 10$

Bonet *et al.* 81

R. Bonet, A. Kung, K. Ripken, R.K. Yakes, M. Sommer & J. Winkler, *Ada Syntax Diagrams for Top-down Analysis*, *SIGPLAN-Notices*, vol.16, n°9, Sept. 1981, p.29-41. $\leftarrow 14$

Boussard & Duby 71

J.C. Boussard, J.J. Duby (éd.) *et al.*, *Rapport d'évaluation d'Algol 68*, R.I.R.O., 5^e année, B-1, 1971, p. 15-106. $\leftarrow 1$

Boute 80

R.T. Boute, *Simplifying Ada by removing limitations*, *SIGPLAN Notices*, vol.15, n°2, 1980, p. 17-28. $\leftarrow 10,12$

Branquart et al. 82

P. Branquart, G. Louis & P. Wodon, *Aspects de Chiff, le langage du CCITT, TSI, vol. 1, n°1, 1982, p. 43-52.* ← *

Brinch-Hansen 75

P. Brinch-Hansen, *The Programming Language Concurrent Pascal, IEEE Transactions on Software Engineering vol.1, n°2, 1975, p. 199-207.* ← 8

Brinch-Hansen 78

P. Brinch-Hansen, *Distributed Processes, Communications of the ACM, vol.21, n°11, 1978, p. 934-941.* ← 8

Brown 77

W.S. Brown, *A Realistic Model of Floating-Point Computation, Mathematical Software III, (J. Rice ed.), Academic Press, New-York, 1977 p. 343-360.* ← 3

Burstall & Goguen 77

R.M. Burstall & J.A. Goguen, *Putting theories together to make specifications, Proceeding of the 5th International Joint Conference on Artificial Intelligence, Cambridge Mass., 1977, p. 1045-1058.* ← 10

Campbell & Habermann 74

R.H. Campbell & A.N. Habermann, *The Specification of Process Synchronization by Peth Expressions, Colloque sur les aspects théoriques et pratiques des systèmes d'exploitation, IRIA, Paris, 1974.* ← 8

Cody 81

W.J. Cody, *Analysis of Proposals for the Floating-Point Standard, Draft R.0 of IEEE Task P754, The IEEE Computer Society, 1981.* ← 3

Cole 81

Stephen N. Cole, *ADA Syntax Cross Reference, SIGPLAN Notices, vol.16, n°3, 1981, p.18-47.* ← 14

Daniel & Ingalls 78

H. Daniel & H. Ingalls, *The Smalltalk-76 Programming System Design and implementation, Proceedings of the 5th annual ACM symposium on Principles of Programming Languages, Tucson, Arizona, SIGPLAN-Notices 1974.* ← 8

Demers & Donahue 79

A. Demers & J. Donahue, *Revised Report on Russel, TR 79-389, Department of Computer Science, Cornell University, 1979.* ← *

DeRemer et al. 81

F. DeRemer, T. Pennello & W.M. McKusman, *Ada Syntax Chart, SIGPLAN-Notices, vol.16, n°9, Septembre 1981, p.48-59.* ← 14

Donahue 79

J. Donahue, *On the Semantics of Data Types, SIAM Journal of Computing, vol.8, n°4, 1979, p. 546-560.* ← 10

Eventoff et al. 80

W. Eventoff, P. Harvey & R.J. Price, *The Rendez-vous and Monitor Concepts : is there an Efficiency Difference?, SIGPLAN Notices, vol 15, n°11, 1980, p. 156-165.* ← 8

Feldman 77

J.A. Feldman, *High Level Language Constructs for Distributed Computing, Proceedings of the 5th Annual IRI Conference, Guidel (J.André, J.P. Benâtre réd.), IRIA, 1977, p. 305-314.* ← 8

Goodenough 75

J.B. Goodenough, *Exception Handling : issues and a proposed notation, Communications of the A.C.M., vol.18, n°12, 1975, p. 683-696.* ← 9

- Goodenough 51**
J. B. Goodenough, *The ADA Compiler Validation Capability*, *IEEE-Computer*, vol 14, n°6, 1981, p. 57-64. — 14
- Gordon et al. 79**
M.J. Gordon, R. Milner & C.P. Wadsworth, Edinburgh LCF, *Lectures Notes in Computer Science*, n°78, 1979. — 10
- Hewitt & Atkinson 79**
C. Hewitt & R.R. Atkinson, *Specifications and Proofs Techniques for Serializers*, *IEEE Transactions on Software Engineering*, vol.1, n°1, 1979, p. 10-23. — 8
- Hoare 78**
C.A.R. Hoare, *Communicating Sequential Processes*, *Communications of the ACM*, vol 21, n°8, 1978, p.666-677. — 3
- Horning et al. 74**
J. J. Horning, H.C. Laver, P.M. Melliar-Smith & B. Randell, *A Program Structure for Error Detection and Recovery in Operating Systems*, *Proc. Int. Symposium IRIA 1974*, (Gelenbe, Kayser red.), *Lectures Notes in Computer Science* n°16, p. 171-187, Springer-Verlag, 1974. — 9
- Ichbiah et al. 76**
J.D. Ichbiah et al., *The System Implementation Language LIS*, *CII Reference Manual*, 1976, Louvain-la-Neuve. — *
- Ichbiah et al. 79**
J.D. Ichbiah et al., *Preliminary Ada Reference Manual*, *SIGPLAN Notices*, vol 14, n°6 Juin 1979. — *
- Jensen & Wirth 74**
K. Jensen & N. Wirth, *Pascal User Manual and Report*, Springer Verlag, Berlin, 1974. — *
- Jones & Liskov 78**
A.K. Jones & B.H. Liskov, *Language extension for expressing constraints on data access*, *Communications of the ACM*, vol.21, n°5, 1978, p.358-367. — 7
- Kanda 78**
A. Kanda, *Data Types as Initial Algebras: A Unification of Scott's and ADJery*, *19th Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, 1978, p. 221-230. — 10
- Kruchten 82**
A. & P. Kruchten (trad.), *Manuel de référence du langage de programmation A.D.A.*, Eyrolles éd., Paris 1982, 240 pages. — *
- Lodgaard 81**
H.F. Lodgaard, *Ada, An Introduction & Ada Reference Manual*, Springer Verlag ed., New York 1981. — *
- Lehmann & Smyth 77**
D.J. Lehmann & M.B. Smyth, *Data Types*, *18th Annual Symposium on Foundations of Computer Science*, IEEE computer Society, 1977, p. 7-12. — 10
- Levin 77**
R. Levin, *Program Structures for Exceptional Exception Handling*, PhD, Carnegie Mellon Jun 1977. — 9
- Liskov et al. 77**
Barbara Liskov, Alan Snyder, Russel Atkinson & Craig Schuffert, *Abstraction Mechanisms in CLU*, *Communications of the ACM*, vol.20, n°6, 1977, p. 564-576. — *
- Lovengreen & Björner 80**
R.H. Lovengreen & D. Björner, *On a Formal Model of the Tasking Conception in Ada*, *SIGPLAN Notices*, vol.15, n°11, 1980, p.213-222. — 8

McLaren 77

M. D. McLaren, Exception Handling in PL1, *SIGPLAN Notices*, vol.12, n°3, 1977, p. 101-104. ← 9

Meillar-Smith & Randell 77

P.M. Meillar-Smith & B. Randell, Software Reliability : The Role of Programmed Exception Handling, *Proc. of ACM Conference on Language Design for Reliable System*, 1977, p. 95-100. ← 9

Meyer 79

B. Meyer, Sur le formalisme dans les spécifications, *Rapport Atelier Logiciel n°23, EDF, Département des méthodes et moyens informatiques, Direction des Etudes et Recherches, HI/3206-01*. ← 13

Milner 77

R. Milner, A Theory of Type Polymorphism in Programming, *Report CSR-9-77, University of Edinburgh*, 1977. ← 10

Mitchell et al. 78

J.C. Mitchell, W. Maybury & R. Sweet, *Mesa Language Manual, Palo Alto Research Center, Février 1978*. ← 9

Moffat 81

D. V. Moffat, Enumeration in Pascal, Ada and Beyond, *SIGPLAN Notices*, vol. 16, n°2, 81, p. 77-82. ← 4

Moore 66

R.E. Moore, *Interval Analysis, Prentice-Hall, Englewood Cliffs, 1966*. ← 3

Nebut 74

J.L. Nebut, Conception d'un système de langages de programmation, *Thèse de docteur ingénieur, Université de Paris VI, 1974*. ← 12

Randell 76

B. Randell, System Structure for Software Fault Tolerance, *IEEE Transactions on Software Engineering*, vol.1, n°2, Juin 1975, p.220-232. ← 9

Rault 79

J. C. Rault, interview de J.D. Ichbiah, *Bulletin de liaison de la recherche en informatique et automatique, IRIA, n° spécial, Juillet 1979*, p. 3-10. ← 1

Robert & Verjus 78

P. Robert & J.P. Verjus, Towards Autonomous Descriptions of Synchronization Modules, *Proc. IFIP Congress, North-Holland, Amsterdam, 1978*, p. 981-987. ← 8

Scott 74

Dana Scott, Data Types as Lattices, *SIAM Journal of Computing*, vol.3, n°3, 1974, p.522-587. ← 10

Scowen & Whichmann 74

R.S. Scowen & B.A. Whichmann, The Definition of Comments in Programming Languages, *Software Practice and Experience*, vol.4, n°2, Avril 1974, p. 181-188. ← 14

Sheil 81

B.A. Sheil, The Psychological Study of Programming, *ACM Computing Surveys*, vol.13, n°1, 1981, p. 101-120. ← 14

Steelman 77

Steelman, *Defense Advanced Research Projects Agency, Arlington, Virginia, 1977*. ← 1

Stevenson 81

D. Stevenson, A Proposed Standard for Binary Floating-Point Arithmetic, *Draft 8.0 of IEEE Task P754, The IEEE Computer Society, 1981*. ← 3

Stoneman 80

Stoneman Environment Requirements, Defense Advanced Research Projects Agency, Arlington, Virginia, 1980. ← 12

Thatcher et al 78

J.M. Thatcher, E.G. Wagner & J. Wright, Data type specification : parameterization and the power of specification techniques, *Proceedings of the SIGACT 10th Annual Symposium on Theory of Computing*, 1978, p. 119-132. ← 10

Thorin 81

M. Thorin, Le langage Ada, manuel complet du langage avec exemples, Eyrolles éd., Paris, 1981. ← 14

Wang & Zalewski 81

L.C. Wang & J. Zalewski, Bibliographie Ada, *BIGRE* n°27, déc.81, p. 3-17.

Wegner 80

Peter Wegner, The ADA Language and Environment, *SIGSOFT*, vol.5 n°2, 80, p.8-14. Traduit en français : Le langage Ada et son « milieu de programmation », *BIGRE* n°20, 1980, p. 3-7
← 1,14

Weish & Lister 81

J. Weish & A. Lister, A Comparative Study of Task Communication in Ada, *Software - Practice and Experience*, vol.11, 1981, p. 257-290. ← 8

Van Wijngaarden et al. 76

A. Van Wijngaarden et al., Revised Report on the Algorithmic Language Algol 68, *Acta Informatica*, vol.5, n°1-3, 1976. ← *

Wirth 77

N. Wirth, Modula : A Programming Language for Modular Multiprogramming, *Software - Practice and Experience*, vol.7, n°1, 1977, p. 3-35. ← *

Wirth 80

N. Wirth, Modula-2, *Berichte des Instituts für Informatik*, n°36, March 1980, ETH Zurich. ← *

Wulf et al. 76

W.A. Wulf, R.L. London & M. Shaw, An introduction to the construction and verification of Alphas Programs, *IEEE-Transactions on Software Engineering*, vol.2, n°3, 1976, p. 253-264.

On consultera aussi les actes des récentes conférences suivantes sur les outils d'aide à la programmation :

Genève 81

Journées francophones sur l'informatique, Outils pour la conception et la production de grands logiciels, Genève, 1981 (ADN, INRIA, CNET, CNRS). ← 12

Grenoble 82

Journées BIGRE, Systèmes intégrés de production de logiciels, *BIGRE*, n°27-28, 1982. ← 12

Rennes 80

Journées BIGRE, Environnements de systèmes, Rennes, 1980. ← 12

Rocquencourt 77

Journées BIGRE, Outils de production de systèmes industriels, Rocquencourt 1977. ← 12

San Diego 81

5th International Conference on Software Engineering, San Diego, 1981 (ACM, IEEE, MBS)
← 12

فهرست

الموضوع	الصفحة
الفصل الأول : مدخل	5
الفصل الثاني : التصريحات والأنواع	12
الفصل الثالث : الأنواع الرقمية	30
الفصل الرابع : الاسماء والتعابير	37
الفصل الخامس : تركيبات المراقبة المتتالية	48
الفصل السادس : التقسيم الى زجل	57
الفصل السابع : المدى وإمكانية الرؤية	73
الفصل الثامن : المهام	89
الفصل التاسع : الشواذ أو الاستثناء	137
الفصل العاشر : الشمولية (النوعية)	151
الفصل الحادي عشر : التصريف المتفصل	173
الفصل الثاني عشر : وسائل التكيف	186
الفصل الثالث عشر : المداخل - المخارج	216
الفصل الرابع عشر : العناصر النحوية ، اللغوية والنصية	257
الفصل الخامس عشر : الخاتمة	277
ملحق : مراجع	280

